



Molymerx

INTASM
Disk

The Instant Assembler

**** DISK ****

**Assembly Language Development System
For the TRS-80**

Written by John Blattner



**DISTRIBUTED IN NEW ZEALAND
by**

MOLYMERX Ltd.

**P.O.Box 60152 Titirangi.
Tel.817-4372**

**DISTRIBUTED IN AUSTRALIA
by**

MOLYMERX Pty.Ltd.

**P.O.Box 900 Gosford. NSW 2250
11 Bourke Ave. Tel.(043) 694-888**

TABLE OF CONTENTS

Directory of the Diskette	2
Introduction	3
Preview of Instant Assembler	3
New Features of Version 2.1	3
What an Assembler Does	4
Part I. The Assembler	6
Section 1. Assembler Commands	6
1.1. Composing and Editing	6
CP, ED and Entering Line Numbers, CC	
1.2. Inserting, Deleting, and Moving	13
IS, DL, DM, MB	
1.3. Listing	14
LC, PC, LL, PL, PR, LI, PI, LE, PE, LS, PS	
1.4. Tape Input/Output	17
WS, VS, RS, WO, WE, RE	
1.5. Disk Input/Output	19
OS, IN, MG, OO, OE, IE	
1.6. Miscellaneous	21
AM, RO, FR, DI, KL, EX, MD	
Section 2. Example of the Assembler in Action	23
Section 3. Inside Instant Assembler	27
Part II. The Debugger	30
Section 4. MicroMind Commands	30
4.1. Stepping, Breakpointing, and Executing	30
SP, XC, BD, RN, BK, RB, SB, JP, CL	
4.2. Register and Memory Display	34
RG, MM, AS, P1, P2	
4.3. Utilities	36
FN, DS, HD, DH	
4.4. Symbolic Disassembly and Transfer	38
SD, AD, IA	
4.5. Tape and Printer Commands	38
TP, VF, DP	
Section 5. Example of MicroMind in Action	40
Section 6. Inside MicroMind	42
Part III. The Linking Loaders	44
Section 7. Linking Loader Commands	44
LD, CL, SY, PM, OO, TP, VF, JP	
Section 8. Example of Linking Loader in Action	49
Appendix 1. Legal Instructions for Instant Assembler 2.1	51
Appendix 2. Summary of Assembler Commands	52
Appendix 3. Source Code Entry	53
Appendix 4. Editing Procedures	54
Appendix 5. Entering Line Numbers and Addresses	55
Appendix 6. Parameter Locations and Meanings	56
Appendix 7. Summary of MicroMind and Linking Loader Commands	57
Appendix 8. Adapting to EDTASM	58
Index	63

DIRECTORY OF THE DISKETTE

Your Instant Assembler diskette contains six command files, whose names and functions are:

- DSKIAS/CMD -- Disk Instant Assembler and debugger package.
- DSKLLB48/CMD -- Bottom-Up Linking Loader for users with 48K RAM.
- DSKLLB32/CMD -- Bottom-Up Linking Loader for users with only 32K RAM.
- DSKLLT/CMD -- Top-Down Linking Loader.
- MICROM/CMD -- Relocatable, stand-alone version of the single stepping debugger, including a printing disassembler.
- IASTRF/CMD -- A three-byte program with the sole purpose of providing a nondestructive reentry to Instant Assembler.

The disk is formatted for the current version of TRSDOS for your machine. If you have two drives, you may put this disk in drive 1 and your TRSDOS compatible operating system in drive 0. To load and execute one of these programs, simply type the name as given above and hit ENTER.

If you have only one drive, you will need to use a different procedure. The Instant Assembler disk has a special structure that will allow you to copy the programs on it to a TRSDOS system disk of your own. To do this, use the following step-by-step procedure:

- 1) Put a current TRSDOS system disk in drive 0 and hit RESET.
- 2) When the DOS READY prompt is displayed, remove the TRSDOS disk, insert the Instant Assembler disk, and hit RESET again.
- 3) The disk should "boot up" with our sign on message. This message will tell you which version of TRSDOS it is designed to work with. If the system disk you are using is not the same type, put the correct system disk in drive 0 and go back to step 1.
- 4) The first program name will also be displayed and you will be asked to put your system disk back in drive 0.
- 5) Put your system disk in drive 0 and enter W to write the program onto your own disk.
- 6) You will then be instructed to put the Instant Assembler disk back in drive 0 to copy the next program. Repeat this sequence until all programs have been copied onto your system disk.

INTRODUCTION

Preview of Instant Assembler

Disk Instant Assembler is a powerful, disk-based assembly system for the TRS-80 Model I or Model III. Its unique design has the object of increasing your productivity as a Z-80 assembly language programmer. Among its dozens of convenient features, the following stand out:

(1) Immediate assembly, and immediate detection of most potential errors, as the lines of symbolic assembly language code are entered.

(2) A compactly encoded source format that provides a 2-1/2 to 1 storage advantage (both in memory and on disk) over the standard source code format. For example, all the source code for all the modules in the Instant Assembler package fits on one 35-track, single density disk; the same source code in standard (EDTASM) format would require three such disks.

(3) Production of independently written, relocatable code modules that can be linked by the Linking Loaders (included in the package).

(4) In-memory assembly and immediate debugging with the built-in debugger, featuring single-stepping with full register displays.

Much thought and hard work have been invested in this new version of the highly acclaimed Instant Assembler to make it especially easy to use -- once you have learned how. Yet, the program is so packed with features that it will take some time to learn to exploit all its strengths. You will find the learning easier if you have had previous experience with an assembler such as EDTASM (the TRS-80 Editor/Assembler). In any case, it is assumed that you have (or will obtain) a table of the Z-80 mnemonic instructions, together with a description of their functions -- information such as that provided in the EDTASM manual. Appendix 1 contains a cryptic (but complete) list of the Z-80 instructions, including the undocumented instructions -- all of which are recognized by Instant Assembler 2.1.

New Features of Version 2.1

Disk Instant Assembler 2.1 is an upgrade of Disk Instant Assembler 1.1 and will accept the source code produced by that earlier version, as well as the source code produced by all versions of the Tape Instant Assembler. Earlier versions of Instant Assembler may not be able to read the source code produced by the 2.1 version, if that source code employs any of the new formatting features of Instant Assembler 2.1.

The two most significant additions in Instant Assembler 2.1 are on-line comments (now allowed) and full editing of source lines. Other noteworthy new features include:

- (1) True listing of decimal and negative operands.
- (2) Character constants.
- (3) Separate listing of internal source code errors.
- (4) Output of EDTASM source to disk.
- (5) Conversion of EDTASM source to Instant Assembler format.
- (6) Merging of Instant Assembler source modules.
- (7) Assembly and disassembly of undocumented Z-80 instructions.
- (8) Disassembly and step-wise debugging that can reference the assembler's symbol table.

If you are familiar with an earlier version of Instant Assembler, you will discover many more new features as you read this manual. Indeed, one of the added features is the manual's appendices, which present in summary or tabular form the most essential information for the successful operation of your Instant Assembler.

What an Assembler Does

The basic task of an assembler is quite simple: to translate symbolic machine code (that is, assembly language) into numeric machine code (hex code). For example, a frequently encountered Z-80 instruction is

```
LD    A,B
```

which has the effect of transferring the contents of the B register to the A register. When assembled (and loaded into memory), this instruction will reside in a single byte somewhere in memory as the pattern of bits

```
0 1 1 1 1 0 0 0
```

or 78H. The assembler's role is to translate the symbolic LD A,B into the numeric 78H. The usefulness of this function is due to the fact that it is not at all difficult for a programmer to learn and remember the exact meanings of several hundred instructions such as LD A,B, but it would be exceedingly tiresome to try to memorize a like number of purely numeric codes.

Many Z-80 instructions refer to memory locations, and some of these locations are likely to change as a program evolves. As an example, suppose that you have written a subroutine that clears a certain buffer, and that in an early assembly of your program this subroutine's entry point is 8000H. Then, the instruction

```
CALL 8000H
```

would clear the buffer. But, a later assembly of the (revised) program may place this subroutine at 8056H, in which case all the CALL 8000H instructions would have to be rewritten.

The solution to the problem of changeable addresses is to give these addresses symbolic names that the assembler (or linking loader) can translate to correct numeric values. In the above example you would give the instruction at the entry point of the subroutine a label -- CLRBUF, let's say. Then, the instruction

```
CALL CLRBUF
```

would clear the buffer, and you wouldn't have to concern yourself about the actual numeric value of CLRBUF.

The use of labels (and symbols to reference the labels) can be extended to the case in which you know perfectly well what the numeric equivalent of a label is, but you want the instruction to be self-documenting. For example,

```
CALL 1C9H
```

will clear the video screen of any Model I or III, because there is a ROM subroutine for this purpose whose entry point is 1C9H. But, it might be easier to divine the purpose of the instruction if it were

```
CALL CLRSCR
```

So, we want to EQUate CLRSCR to 1C9H. Z-80 assemblers furnish a "pseudo-op" for this purpose: EQU. The pseudo-instruction we want is thus

```
CLRSCR EQU 1C9H
```

Besides the EQU, Z-80 assemblers provide four other useful pseudo-ops: DEFB, DEFW, DEFM, DEFS. DEFB is used to assemble a one-byte constant into a program; examples are:

```
TWO      DEFB 2      ;Note the label
          DEFB 'X'    ;A character constant
```

DEFW is used similarly to assemble a two-byte constant (but not two character bytes). DEFM is for assembling a string of ASCII characters, as in

```
MESSAG DEFM 'Do not touch the break key.'
```

(Note that apostrophes are required to start and end the string.) Finally, DEFS reserves a block of storage for use by the program; for example

```
BUFFER DEFS 256
```

will reserve 256 bytes, the first of which will be at (symbolic) memory location BUFFER.

The above describes the fundamental duties of your Instant Assembler. All other features are frills designed to save you time in the creation of perfect programs.

PART I. THE ASSEMBLER

The assembler proper is the principal component of the assembler-debugger package that is loaded under the name DSKIAS/CMD. Learning to use the assembler requires first an understanding of its command structure. There are 37 two-letter commands, which will be fully explained in Section 1 below; they have been divided into six subsets for clarity. (These commands are also summarized in Appendix 2.)

SECTION 1. ASSEMBLER COMMANDS

When you load and run Disk Instant Assembler, you see a "?" (and a blinking cursor) displayed at the left side of the screen just below the program title line. This is the prompt for entering an assembler command. The universal rule for the entry of any command (in the assembler, debugger, or linking loaders) is that the entry is completed with the typing of the second letter; the ENTER key does not have to be pressed to enter the command. If you type an unrecognizable command, Instant Assembler will ask for the command again with another prompt.

1.1. Composing and Editing

At the heart of any assembler are the routines that make possible the entering and editing of source (symbolic assembly language) code. If you have not programmed in assembly language before, you are in for a pleasant surprise -- typing Z-80 source code (especially with Instant Assembler) is far easier than typing BASIC code. And, if you have found the editing of lines of BASIC a slow and frustrating task, you should be delighted with the editing facilities of Instant Assembler 2.1.

CP (ComPose)

Enter "CP" in response to the "?" prompt to commence the composition of an assembly language program. If the source buffer is not empty, Instant Assembler will respond "CODE ERASURE. PROCEED (Y/N)?" Either type "Y" to erase the buffer and to proceed with composing, or else press "N" (or almost any other key) to abort the CP command. (Instant Assembler has several protective features such as this one; they have been tailored to be as unobtrusive as possible.)

If the source buffer is empty (as it will be when you first load the program), and you enter the CP command, you will be given a blank line number 1 on which to start your program. You will also notice a vertical bar followed by the letter X in the lower right corner of the screen. The reason for this display is to remind you that you are in the "X" mode, which is the normal mode for entering source code. It is possible to leave the X mode for the purpose of editing the line that you are entering; how to do this will be explained along with the ED (EDit) command a bit later. Now here are the rules for entering source code; it is recommended that you learn them through practice rather than by memorization.

(1) A composition line normally has three fields: label, opcode, operand. Use the RIGHT ARROW key to tab to the next field; you cannot tab farther than the operand field. Use the LEFT ARROW key to backspace and erase the previous character, including backspacing to the previous field if necessary. Each field is restricted with respect to the number and type of characters that it will accept:

The label field accepts only a letter or the ampersand (&) as its first character, only letters or digits for its subsequent characters, with a maximum of six characters.

The opcode field accepts only letters, with a maximum of four characters.

The operand field accepts anything (including spaces), with a maximum of 45 characters.

It is not possible to enter more characters in a field than the field limits just given. The ampersand as the first character of the label field is used to designate an external label -- one whose value can be made available to other modules by the Linking Loader.

(The information of the above paragraphs is presented in more graphic form in Appendix 3.)

(2) An on-line comment (that is, a comment that appears on the same line as an instruction) is entered in the operand field, following the operand (or operands), and preceded by a semicolon (;). One or more spaces may precede the semicolon, but they are not required. Examples of correctly entered on-line comments are:

```
EXIT1  CALL  60H          ;Time delay
        JP    402DH       ;Return to DOS
```

With the Model III, comments may be entered in lower case; use the lower case (SHIFT-0) toggle, but be sure to restore the upper case mode when the comment is completed. Don't worry about alignment of comments when entering source lines -- they will be aligned automatically in listings.

(3) No on-line comment is allowed with a DEFM pseudo-instruction. This is not a severe restriction, since the DEFM is usually self-documenting.

(4) By entering a semicolon as the first character of a line, you override the three-field format given in (1) and convert the entire line to a comment line, with a maximum of 59 characters (including the semicolon). (This initial semicolon cannot subsequently be erased with the LEFT ARROW key; if erasure is necessary, use SHIFT-LEFT ARROW.)

(5) Instant Assembler does not recognize the ORG, END, or DEFL pseudo-ops. ORG and END are supplied automatically when a program is listed or recorded. (You may use the RO command, explained in subsection 1.6, to set or change the origin.)

(6) Symbols follow the rules for labels -- six characters maximum, first character either a letter or ampersand, subsequent characters either letters or digits. Symbols may be postfixed with decimal offsets in the range of -31 to +287, inclusive. (An offset gives the number of bytes of displacement, just as in EDTASM.) Any other combinations involving symbols are not legal; thus, LD HL,HOLD+42 is allowed, but LD HL,HOLD-42 and LD HL,HOLD2-HOLD1 are not. A symbol may represent an address or a 16-bit constant, but may not be used for an 8-bit constant; thus, JR THERE is valid, but LD A,SPACE is not.

(7) Numeric constants may be entered in either decimal or hexadecimal -- though the first character must be a digit -- and may be prefixed with a minus sign. Hex constants and addresses must also bear the postfix "H". Legal entries include:

LD	A,0CFH	LD	HL,23586
LD	B,-5	LD	DE,-9
INC	(IX-0BH)	LD	BC,0A5A5H
ADD	A,(IX+20)	CALL	33AH

Note particularly that the increment (or decrement) to an index register may be in either decimal or hex and must lie in the range of -128 (-80H) to +127 (+7FH), inclusive.

(8) Character constants may be used for 8-bit operands. A character constant must be preceded and followed by an apostrophe (single quote mark). Examples:

```
LD    A,'X'
CP    ';
```

(9) The pseudo-ops DEFB, DEFW, DEFM, and DEFS may be entered economically by using SHIFT-1 (!), SHIFT-2 ("), SHIFT-3 (#), and SHIFT-4 (\$), respectively. This may be done either from the label field or in the first character position of the opcode field. For example, SHIFT-1 from the label field is equivalent to typing the sequence TAB (that is, RIGHT ARROW), "DEFB", TAB, and places you immediately in the operand field for entering the value of the byte. This feature is provided for convenience in assembling tables, messages, and storage areas.

(10) The operand for a DEFS pseudo-op is restricted to the range 1 to 4095 (decimal), inclusive. To reserve more than 4095 bytes of storage, use multiple DEFS's. DEFS 0 is illegal.

(11) Because of the 45 character limit in the operand field, a DEFM pseudo-op cannot define a string of more than 43 characters, since an apostrophe (single quote) is required both to begin and to end the string. To define a longer string, use multiple DEFM's. With the Model III, you may enter lower case characters in a DEFM string; just be sure to return to the upper case mode when the entry has been completed.

(12) The operand for an EQU pseudo-op must be an absolute address. As examples, HERE EQU 823BH is legal, but HERE EQU THERE+1 is not. Note also that an EQU must have a label.

(13) All relative jumps (JR, JR NZ, JR Z, JR NC, JR C, DJNZ) must refer to symbolic target addresses. Examples of legal relative jump instructions are:

```
DJNZ  LOOP
JR    NZ,DELAY-3
```

Any relative jump to an absolute address will be rejected; also, Instant Assembler does not recognize "\$" as a reference to the memory location of the present instruction. Hence, the following are illegal:

```
JR    5024H
JR    C,$-12
```

(14) If you make a real mess in entering a line and would like to have a fresh start, type SHIFT-LEFT ARROW. You will then get a blank line with the same line number.

(15) When a line of source code is complete, enter it by pressing ENTER. Instant Assembler will immediately assemble it (except for a possible reference to an as-yet-undefined label). If there is no detectable error, the instruction is accepted and you are presented with the next line number in sequence for continuation. To end composition, press the BREAK key.

(16) If any error is detected in an entered line of source code, Instant Assembler will announce it with a message. Possible error messages at this stage are:

```
MISSING LABEL
ILLEGAL LABEL  -- (Label is a Z-80 operand, such as "HL".)
DBLY DFND LABEL -- (Label has been used before.)
MISSING OPCODE
ILLEGAL OPCODE -- (Not a Z-80 opcode.)
MISSING OPERAND
ILLEGAL OPERAND -- (Not a Z-80 operand.)
BAD OPERAND     -- (Many possible reasons, including field overflow,
                  incorrect punctuation, and improper mixing of operands.)
OUT OF RNG      -- (Backward relative jump is too long.)
```

Following display of the message, Instant Assembler switches to change (edit) mode for correction of the line and positions the blinking cursor in the offending field. How to make the correction will be explained under the ED command, which comes next.

ED (EDit)

The ED command allows you to inspect and change as many consecutive lines of source code as you please in one continuous operation; insertions and deletions of lines may also be freely intermixed with the changes. To understand the workings of this command, it is helpful to identify three separate levels at which activities take place. For want of better names, let us call these levels the "line" level, the "cursor" level, and the "edit" level. At the line level, a line of source code is displayed for your inspection, but there is no cursor; Instant Assembler awaits your instructions for the disposition of this line. If you choose to descend to the cursor level, a nondestructive blinking cursor appears that can be freely moved about without changing any characters in the field; most ordinary characters typed in at this level are simply ignored. Finally, you can descend to the edit level, where characters that you type are entered into the source line. Of course, it is possible to move upward in the level hierarchy, too.

After you have entered the command "ED" in response to the "?" prompt, you will be asked for a "FIRST LINE#?". Type the number of the first line that you wish to edit (or inspect), and press ENTER. This line of source code will then be displayed. You are now at the line level, and you have several options for the disposition of the displayed line:

UP ARROW -- Press this key to back up one line. The previous line is displayed, and you remain at the line level.
 DOWN ARROW -- Press this key to advance one line. The next line is displayed, and you remain at the line level.
 ENTER -- Same as DOWN ARROW.
 D -- Press the D key to Delete the displayed line. The next line is then displayed (with the same line number as the deleted line), and you remain at the line level.
 I -- Press the I key to Insert a line just before the displayed line. Descent is to the edit level (and the "X" mode) for entering the new line.
 C -- Press the C key to Change (edit) the displayed line. Descent is to the cursor level.
 BREAK -- Exit to Instant Assembler command level.

To clarify the action of the I key at line level, suppose that the displayed line is line number 237. When you press the I key, you will be given blank line number 237 on which to compose the line to be inserted. After this line has been typed and entered (exactly as described under the CP command), the original line number 237 will be displayed again with its line number changed to 238. (And all following lines will have their line numbers increased by 1 because of the insertion.) You will be back at the line level, and you can, if you wish, use the I key again and again to insert any number of lines ahead of the original line 237.

If you want to edit the displayed line, use the "C" key to descend to the cursor level. At this level you have available several cursor motion commands:

SPACE -- Move cursor one space to the right without erasing the character.
 LEFT ARROW -- Move cursor one space to the left without erasing the character; move to previous field if cursor is at left end of present field.
 RIGHT ARROW -- Tab to the next field.
 n SPACE -- Move cursor n spaces to the right without erasing.
 n LEFT ARROW -- Move cursor n spaces to the left without erasing.

In the above, n represents a one- or two-digit number that you have typed before the SPACE or LEFT ARROW; if you type more than two digits, only the last two will be used. Also, the movement will not go beyond either end of the present field (except when n LEFT ARROW is used at the left end of a field). Once you have positioned the cursor where you want it, you may descend to edit level in any one of the following ways:

SHIFT-D -- Delete the character at the cursor, and return to cursor level.
 n SHIFT-D -- Delete n characters starting with the one at the cursor, but not extending beyond the present field. Return to cursor level.
 SHIFT-I -- Insert characters in front of the character at the cursor.
 SHIFT-C -- Change (retype) characters starting at the cursor.
 SHIFT-H -- Hack and enter; that is, delete characters from the cursor to the end of the field, and then go into entry mode.
 SHIFT-X -- Move cursor to right end of present field and go into entry mode.

These editing modes (as well as the cursor motion commands) are nearly identical to the ones in the LEVEL II BASIC line editor, except for the extra SHIFT required to initiate some of them. In the entry mode (after SHIFT-H or SHIFT-X), the LEFT ARROW erases characters as it backspaces, and the SPACE (if it is allowed in the field) also erases characters. In the I and C modes, however, the LEFT ARROW does not erase characters. The I, C, H, and X modes are continuous; that is, you remain in these modes until you either tab or backspace to another field, or until you cancel the modes with one of these keys:

DOWN ARROW -- Cancel I, C, H, or X mode and return to cursor level.
SHIFT-UP ARROW -- Same as DOWN ARROW.

As you use the various edit modes, the display at the bottom right corner of the screen (a vertical bar followed by a blank or a character) will change to reflect the mode that you are in. Digits typed at the cursor level will also appear in this window; the character M is used to signify any two-digit entry larger than nine. If you are observant, you may notice a couple of apparent anomalies that are, however, deliberate and correct: X mode turns into H mode when you backspace, and Instant Assembler itself initiates the X (or H) mode whenever you tab (forward) to an empty field or backspace to a previous field. A little thought will convince you that the X (or H) mode is usually the desired one in these circumstances.

If you botch the editing of a line, you can type SHIFT-LEFT ARROW to get a blank line with the same line number for reentering the code.

When a line has been edited to your satisfaction, press the ENTER key. The revised line will then be checked for errors; if one is found, it is reported, and you are returned to cursor level with the cursor positioned in the offending field. When the corrected line is finally accepted, Instant Assembler returns to line level and displays the next line of source code, except that, if ED is used all the way to the end of the source buffer, an automatic exit is then made to the Instant Assembler command level.

When you descend from the line level to the cursor level by pressing the "C" key, the displayed line is immediately deleted from the source buffer to make way for the revised line. Since it would be untidy to leave this unintended hole in the source code, Instant Assembler will not release you from the ED command until you have entered an error-free version of the edited line. For this reason, the BREAK key functions exactly like the ENTER key here, rather than effecting a return to the command level.

Though the ED command has now been fully explained, we are not finished. The same editing facilities are sometimes used in connection with the CP command, but the details of initiation and termination are somewhat different.

When you enter source code (with the CP command), you are normally in the X (or H) mode. (Thus, with the CP command, you enter the editing process at the bottom, or edit, level.) If you decide that you would like to change part of what you have entered, use the DOWN ARROW key to ascend to cursor mode, position the cursor, and make the change. Return to where you were working with the TAB (RIGHT ARROW) and/or SHIFT-X keys.

NOTE: The keyboard might seem to die if you inadvertently hit the DOWN ARROW key while in X (or H) mode. If this happens, merely type SHIFT-X to restore normal operation.

A line of source code that you have composed with the CP command may have an error that Instant Assembler detects. In this case, you are thrust into the editing process at the cursor level, with the cursor positioned in the offending field. (Review item (16) under the CP command.) The editing procedures are still the same, of course. Note, however, that you can use the BREAK key here to escape to the command level, since no line of source code has been deleted by the editing procedures of CP mode.

(Editing procedures are summarized in Appendix 4.)

Entering Line Numbers

The ED command and several of the commands to be described later require a starting line number. (Some of the others also require an ending line number.) Since Instant Assembler prompts for the information that it needs, you do not have to remember any special syntax (such as "ED:122", or "ED,122"). Also, for your convenience in finding lines, Instant Assembler provides three different ways in which line numbers can be entered:

(1) As decimal numbers. This direct method requires that you know your targeted lines by number. The listing commands (Section 1.3) can be helpful in finding these line numbers.

(2) As labels (with optional decimal offsets in the range of -31 to +99, inclusive). For example, asking for the line "EXIT" would cause Instant Assembler to find the line of source code with the label "EXIT". Asking for "EXIT+10" would direct Instant Assembler to the line whose line number is 10 larger than that of the line with label "EXIT". Note that the offset here is the number of lines of offset from the specified label. This feature of being able to address a line by its label (or label plus offset) makes it easy to find lines in a large program if you have a rough (hand-written) copy of the source code, or a printed listing of an earlier version of the program.

(3) By means of the current line pointer. Instant Assembler maintains a current line pointer that contains the line number of the last line to have been displayed on the screen (or, in some cases, the line before that one). When a line number is requested, you may use the "." (period) key to demand the current line. If you press the UP ARROW (or the DOWN ARROW) key, you request the line before (or the line after) the current line.

If you use method (1) or (2) above, and if you enter a line number that is less than 1 or larger than that of the last line in the source buffer, Instant Assembler will respond "BAD" and ask for the line number again. The same is true if you enter a label (or label plus offset) that does not correspond to any line of source code.

(This material on entering line numbers is summarized in Appendix 5.)

CC (Continue Composition)

After composition has been ended with the BREAK key, it may be continued by entering the CC command. You will be given a line number one larger than that of the last line of code in the source buffer to continue your program. CC may also be used to add to a source program that has been read in from disk or tape. An auxiliary use of the CC command is to find the line number of the last source line; when the CC command is entered, the current line pointer is set to the number of this last line.

1.2. Inserting, Deleting, Moving

These commands all require a starting line number; two of them (DM and MB) also require an ending line number, and MB requires yet a third line number. Refer to the paragraphs on Entering Line Numbers in Section 1.1 (at the end of the ED command).

IS (InSert)

When you use the IS command, Instant Assembler will ask for a "LINE#?". The insertion will be immediately before the line whose number you specify. For example, if you insert at LINE# 69, the inserted line will then have the line number 69, while previous line 69 will become line 70, previous line 70 will become line 71, etc. After you have composed the new line (exactly as with the CP command), it is inserted, and Instant Assembler will give you the next line number for continued insertion. You may insert as many instructions as you please. Use the BREAK key to exit from IS mode.

DL (Delete Line)

Use DL to delete a single line. Instant Assembler will ask for the "LINE#?". The deleted line will be displayed on the screen to confirm the correctness of the deletion. The line numbers of all lines following the deleted line will be decreased by 1.

DM (Delete Multiple lines)

Use DM to delete a block of lines. Instant Assembler will ask for the "FIRST LINE#?" and the "FINAL LINE#?" of the block in two separate questions. These may be independently entered as decimal numbers, or labels plus offsets, or with the current line pointer facility. The first line of the deleted block will be displayed on the screen as a partial confirmation of the correctness of the deletion. Multiple deletion reduces the line numbers of all lines following the deleted block.

MB (Move Block)

Use MB to move a block of source code from one position to another. Instant Assembler will ask for three line numbers (with separate queries). "FIRST LINE#?" and "FINAL LINE#?" designate the first and last lines of the block to be moved, while "INSRT LINE#?" is the line number at which the block will be inserted. (It will be inserted just ahead of the line whose line number is the INSRT LINE#.) The INSRT LINE# must either be less than the FIRST LINE# or greater than the FINAL LINE# plus one; otherwise, you will get a "BAD" message and a request for reentry of this line number. MB will obviously have a drastic effect on many line numbers.

To move a block to the end of the program, first add a NOP at the end (using the CC command), move the block to just in front of the NOP (INSRT LINE# = ".", using the current line pointer facility), then delete the NOP.

1.3. Listing

In the listing commands, a first letter of "L" directs the listing to the screen, while a first letter of "P" directs the listing to the line printer. (If printer output is selected, have the printer turned on and ready.) The LL, PL, and PR commands require one or two line numbers; enter these as explained in Section 1.1 (at the end of the ED command).

For every screen listing command except LI, 12 (or, sometimes, 13) lines are presented at a time for your inspection; when you are ready for the next 12 lines, press ENTER (or any key except BREAK, SPACE BAR, or UP ARROW). This 12-lines-at-a-time progress of screen listings may be overridden by depressing the SPACE BAR. Holding the SPACE BAR down will cause continuous scrolling of the listing; this scrolling will stop instantly when you release the SPACE BAR. Rapid depression and release of the SPACE BAR will effect the listing of one or two additional lines of the program. With the SPACE BAR released, ENTER will act in its usual fashion to cause the listing of another 12 lines. When using the LC or LL command, after a pause in the video listing, the UP ARROW key will cause the listing to move backward about 10 lines, so that you can review it. Thus, the ENTER, SPACE BAR, and UP ARROW keys give you pin-point control of listings to the screen.

In a listing to the line printer, Instant Assembler indents each line eight spaces to provide a left margin for binding. After each 59 lines of printed listing, Instant Assembler supplies seven line feeds for pagination in the standard (66 lines per page) printer format. The 59-counter is reset to zero each time a new listing command is entered. The print parameters given here (number of spaces of indentation, number of lines per page), and a few others, can be changed so as to produce printed output in almost any format that you want; how to do so is explained in Appendix 6.

Any listing can be terminated by depressing the BREAK key and holding it down for a bit.

All source code listings are assembly listings; the format is essentially that of EDTASM, with the following exceptions:

(a) Bytes of hex code are separated by spaces for readability. (b) The first four assembled bytes for a DEFM instruction are displayed as for other instructions. Bytes after the fourth are not listed. (c) The memory addresses of all instructions except EQU's are shown at the extreme left of the listing. No memory address is displayed for an EQU pseudo-instruction.

(It is also possible to obtain source-only printed listings by changing certain print parameters. See Appendix 6 for directions.)

Besides the screen listing commands described below, Instant Assembler allows a quick listing of the current line. From the command level, press the PERIOD key to display the current line; the UP ARROW and DOWN ARROW keys function similarly to list either the line before, or the line after, the current line. This current line listing facility also has an extension: By holding down the SPACE BAR before pressing the PERIOD, UP ARROW, or DOWN ARROW, you can cause 14 lines to be listed, ending with the current line, its predecessor, or its successor.

LC (List Completely)

LC causes a complete listing to be posted to the screen (with a pause after each 12 lines unless the SPACE BAR is held down). A complete listing consists of the ORG line (supplied by Instant Assembler), the assembled source code (with an error message at the right end of each line in error), the END line (also supplied by Instant Assembler), the error count, and the symbol table. The symbol table is in alphabetic order and is printed four symbols to a line for compactness. The possible error messages in a listing are just these two:

****OOR**** (Out Of Range -- relative jump is too long.)
****UDS**** (UnDefined Symbol.)

(A source line may require two lines of listing. In this case, if an error message is also required, it will appear at the right end of the second line.)

NOTE: If you use the UP ARROW key with the LC command to review the listing, errors that are passed over will be counted again in the forward listing, so that the final error count may be too large.

PC (Print Completely)

PC is like LC, except that the output is to the line printer.

LL (List to the Last line)

After the LL command is entered, Instant Assembler will ask for a "FIRST LINE#?". Respond with the line number (as a decimal, label plus offset, or current line) of the first line to be listed. The listing will commence there and continue (with a pause after each 12 lines) to the last source line, or until the BREAK key is used to terminate the LL command. No symbol table will be listed.

PL (Print to the Last line)

PL is like LL, except that the output is to the line printer.

PR (Print a Range of lines)

After the PR command is entered, Instant Assembler will ask for a "FIRST LINE#?" and a "FINAL LINE#?". These are the numbers of the first and last lines of source code to be listed, and may be entered independently as decimal number, label plus offset, or current line. Output is to the line printer.

LI (List Internal errors)

All ****OOR**** (Out Of Range) errors, all undefined internal symbols (those not commencing with "&"), and all relative jumps to undefined external symbols are internal errors. It is essential to correct a module's internal errors before employing the Linking Loader to link it to other modules. The LI command makes it easy to find and correct all internal errors in your program.

When the LI command is entered, Instant Assembler will display the first source line that has an internal error and then pause. (This pause is at the "line" level, as described under the ED command.) You now have three choices:

(a) BREAK will return you to Instant Assembler command level. (b) Either DOWN ARROW or ENTER will cause Instant Assembler to find and display the next source line that has an internal error. (c) Pressing the "C" key opens the displayed line for editing -- exactly as with the ED command. (The "C" key causes a descent to the cursor level.) When the editing is completed (by pressing ENTER), and the edited line is accepted, Instant Assembler will find and display the next source line that has an internal error.

After the last line with an internal error has been disposed of, the count of internal errors will be displayed; this count will be zero if there were no such errors to begin with.

PI (Print Internal errors)

The PI command causes all source lines with internal errors to be listed on the line printer, with a count at the end.

LE (List External undefined symbols)

LE causes all external undefined symbols to be listed to the screen; they are listed in alphabetic order, eight to a line. Each symbol is listed only once, even if it occurs several times in your program.

External undefined symbols may not be actual errors, since they may correspond to labels in other modules. The LE command allows you to check that no typographical errors have been made in these symbols. If real errors are discovered, the FR command (Section 1.6) can be used to locate and correct them painlessly.

PE (Print External undefined symbols)

PE is like LE, except that the output is to the line printer.

LS (List Symbols)

LS causes the symbol table to be listed to the screen. This listing is in alphabetic order, four symbols to a line. (External symbols are listed first.) The defined value of each symbol is displayed next to the symbol. (Actually, what is listed is a label table. Undefined symbols are not listed.)

PS (Print Symbols)

PS is like LS, except that the output is to the line printer.

1.4. Tape Input/Output

In all its tape input/output functions Instant Assembler prompts for the items of information (titles, addresses) that it needs. Any tape command can be aborted by pressing the BREAK key in response to a request for information. In the Model III, all tape commands of Instant Assembler set the cassette speed to 500 baud to ensure reliable recordings and to allow necessary processing during loading.

WS (Write Source)

The WS command is used to record a source tape (in Instant Assembler format) of the program in the source buffer. After the WS command is entered, you will be asked for a "TITLE?". The title is restricted to 6 characters, the first of which must be a letter; subsequent characters must be either letters or digits. Have the tape ready for recording, with the PLAY and RECORD keys of the cassette depressed. As soon as you press ENTER after typing the title, the recording will begin.

VS (Verify Source)

After recording an Instant Assembler source tape with the WS command, rewind the tape and use the VS command to verify it. (Have the tape ready for reading before entering the VS command.) VS requires no arguments and returns either "GOOD" or "BAD" in reporting on the verification. In case of a "BAD" verify, try adjusting the volume before repeating the VS command; as a last resort, record the program again and verify it.

RS (Read Source)

The RS command causes a source tape (recorded in Instant Assembler format with the WS command) to be read into the source buffer for editing, assembling, or debugging. (This source code will replace any that is already in the buffer.) If the source buffer is empty when you type the RS command, Instant Assembler will ask for a "TITLE?"; when this has been entered, the tape will be read. If the source buffer is not empty when you enter the RS command, Instant Assembler will respond "CODE ERASURE. PROCEED (Y/N)?". If you decide to proceed with the source input, type "Y"; you will then be asked for a "TITLE?". (The title is restricted to six characters, the first of which must be a letter.) Have the tape ready for reading as you complete the entry of the title. Instant Assembler will report on the read with either a "GOOD" or a "BAD" message. In case of a "BAD" read, rewind the tape and try again (perhaps adjusting the volume before the second try.)

WO (Write Object)

The WO command is used to record an object tape (in SYSTEM format) of the program in the source buffer. After the WO command is entered, you will be asked for a "TITLE?", an "ORIGIN?", and an "ENTRY ADDRESS?". (The entry address is the point at which the program will be entered after a SYSTEM load and a response of "/" to the following "*" prompt.) When the entry address has been entered, recording will commence. (Caution: have the cassette ready for recording, for the completion of this entry may not require pressing the ENTER key.)

The origin and entry addresses may be independently entered in any of the following ways:

(a) By default to the value of the source code origin, as set with the R0 command and as displayed at the beginning of a listing with the LC command. To request this default value, merely press the ENTER key in response to the address query. (b) As a hexadecimal address. For this mode, enter the address as four (or fewer) hexadecimal digits. Do not enter a zero in front of a leading A, B, C, D, E, or F. Do not type "H" at the end of the entry. (c) As a decimal address. For this mode, enter five (no fewer) decimal digits. Pad with leading zeroes to make up the required five digits. (This will not usually be necessary, since nearly all programs will have origins above 10000 decimal.)

(Methods (b) and (c) are used throughout the Instant Assembler package for the entry of addresses. Appendix 5 repeats this information.)

NOTE: Object code recorded with the W0 command is in one contiguous block; there are no skips for DEFS pseudo-ops. In fact, a DEFS instruction causes the specified number of bytes to be recorded as zeroes on the object tape.

WE (Write Edtasm source tape)

The WE command is for recording a source tape (of the program in the source buffer) that can be read and edited by EDTASM. You will be asked for a "TITLE?" and an "ORIGIN?"; recording commences as soon as the latter is entered. (The origin can be entered in any of the three ways described under the W0 command.) The line numbers for a source tape produced with the WE command start with 00000 (for the ORG line) and proceed in steps of 10.

RE (Read Edtasm source, translate, and merge)

The RE command allows you to translate EDTASM source tape to Instant Assembler format. You will be asked for a "TITLE?"; after this has been entered, the entire EDTASM tape will be read into RAM above Instant Assembler's source buffer. (If this code is too extensive to fit into your memory, reading will halt with an "OUT OF MEM" report.) With the EDTASM source in memory, translation commences; each source line is displayed on the screen, translated, and added to Instant Assembler's source buffer. If an error is detected in a line, translation is interrupted, an error message is posted, and a blinking cursor appears in the offending field. (You are at the cursor level in the editing process.) Make the necessary correction in the line, press ENTER, and the translation and merging will continue.

NOTES: (1) The new source code is added to any code already in the source buffer; if you want to clear out the buffer before translating an EDTASM tape, use the CP command, answer "Y" to the "CODE ERASURE. PROCEED (Y/N)?" query, then press BREAK.

(2) Use of the BREAK key at any time in the translation process (including during an edit) will terminate the operation. If you use the SHIFT-LEFT ARROW while editing a line, that line will be deleted from the translated source code.

(3) The ORG and END lines of the EDTASM source are translated as comment lines in the Instant Assembler source.

(4) Since some EDTASM lines do not permit direct translation to Instant Assembler format, a bit of ingenuity will occasionally be required to complete the operation. For a tough problem like LD BC,END-BEG+1, you may have to make a temporary change (to allow the translation to proceed), note where this line occurs, and return to it for a more conscientious edit when the translation has been completed. (For a number of suggestions on how to make these edits, refer to Appendix 8.)

1.5. Disk Input/Output

In all its disk input/output functions Instant Assembler prompts for the items of information (file names, addresses) that it needs; it also provides protective mechanisms to minimize the chance of inadvertent erasure of either a disk file or the source buffer. Any disk command can be aborted by pressing the BREAK key in response to a request for information. A blinking asterisk appears in the upper right corner of the screen during disk transfers. If any error occurs, the diagnostic message supplied by DOS is displayed, and Instant Assembler then allows you either to repeat or abort the operation.

OS (Output Source to disk)

To save source code (in Instant Assembler format) on disk, enter "OS" in response to the "?" prompt. You will then be asked for a "FILE NAME?". Enter this name in standard file specification format, including extension and drive number. (It is a good practice to designate your source code files with their own reserved extension, such as "SRC".) After the file name has been entered, DOS will be requested to open the file. If no file with this name exists on the disk, the file will then be initialized, the legend "NEW FILE." will be displayed for your information, and the source code will be recorded and verified.

If a file already exists on the disk with the file name that you entered, Instant Assembler will respond, "FILE REWRITE. PROCEED (Y/N)?". Type "Y" here to proceed with rewriting this file, or else type "N" (or almost any other character) to abort the OS command.

IN (INput source code from disk)

The IN command causes a source file (previously recorded using the OS command) to be read from disk and placed in the source buffer for editing, assembling, or debugging. (This source code will replace any that is already in the buffer.) If the source buffer is empty when you enter the IN command, Instant Assembler will ask you for a "FILE NAME?" and then transfer the source code from this disk file. If the source buffer is not empty when you enter the IN command, Instant Assembler will respond "CODE ERASURE. PROCEED (Y/N)?"; you then have an obvious choice of proceeding with or aborting the input request. If you proceed, the source buffer will be erased, and you will then be asked for a "FILE NAME?".

MG (MerGe source code from disk)

The MG command allows you to merge Instant Assembler source modules. You will be asked for a "FILE NAME?"; after this has been entered, the entire Instant Assembler source file will be read into RAM above the source buffer. (If this code is too extensive to fit into your memory, reading will halt with an "OUT OF MEM" report.) With the new source file in memory, merging commences; each line is displayed on the screen and added to Instant Assembler's source buffer. If a detectable error is encountered, merging is interrupted, an error message is posted, and a blinking cursor appears in the offending field. Edit the error, make a note of where it was, press ENTER, and the merging will continue. (An error in merging is either a doubly defined label or an out of range relative jump to an earlier label that is about to be doubly defined. Change the second occurrence of the label, then, after the merging is complete, you may use the FR command to find all references to the old label and change the appropriate ones among these to refer to the new label.) The BREAK key may be used at any time to terminate the merge operation. SHIFT-LEFT ARROW, used on a line with an error, will delete that line.

OO (Output Object code to disk)

The OO command is used to record (on disk) an assembled version of the program in the source buffer. This recorded program is in standard disk object format, ready to be loaded and executed from DOS. To use the command, type "OO" in response to the "?" prompt. You will then be asked for a "FILE NAME?", an "ORIGIN?", and an "ENTRY ADDRESS?". The origin and entry address may be entered in any of the three ways that are given under the WO command (Section 1.4.) and in Appendix 5. When all this information has been entered, DOS will be requested to open the file. Depending upon the outcome of this request, Instant Assembler will report "NEW FILE." (followed by transfer of the object code to disk) or "FILE REWRITE. PROCEED (Y/N)?". In the latter case you then have a choice of continuing or aborting the operation.

NOTE: Object code recorded with the OO command is in one contiguous block; there are no skips for DEFS pseudo-ops. In fact, a DEFS instruction causes the specified number of bytes to be recorded as zeroes in the object file.

OE (Output Edtasm source to disk)

The OE command is for recording (on disk) a source file that can be read and edited by disk EDTASM. After entering this command, you will be asked for a "FILE NAME?" and an "ORIGIN?". Enter the origin in any of the three ways given under the WO command and in Appendix 5. When this information has been entered, DOS will be requested to open the file. Depending upon the outcome of this request, Instant Assembler will report "NEW FILE." (followed by transfer of the source file to disk) or "FILE REWRITE. PROCEED (Y/N)?". In the latter case you then have a choice of continuing or aborting the operation.

The line numbers for an EDTASM source file produced with the WE command start with 00000 (for the ORG line) and proceed in steps of 10. The source file is normally recorded with six initial ASCII spaces (a dummy title), which seems to be the format that is expected by most disk versions of EDTASM. It is possible, however, to suppress these six characters in the recording if your disk EDTASM doesn't accept them; how to do so is explained in Appendix 6.

IE (Input Edtasm source, translate, and merge)

The IE command functions exactly like the RE command (Section 1.4), except that you are asked for a "FILE NAME?" (instead of a "TITLE?"), and the input is from disk. Refer to the RE command for a complete description of this operation.

While Instant Assembler is a complete assembly system, the OE and IE commands have been provided so that you may use it in conjunction with EDTASM.

1.6. Miscellaneous

AM (Assemble-to-Memory)

The AM command permits you to assemble a source program directly into memory. Once assembled, the program may be debugged with the debugging subsystem (MicroMind). After the AM command has been entered, Instant Assembler will respond "1ST FREE MEM: XXXX", where the XXXX is the hexadecimal address of the first memory location available for the assembly. You will then be asked for an "ORIGIN?", which may be entered in any of the three ways that are given under the WO command (Section 1.4.) and in Appendix 5. This origin must be at least as high as the number announced in the 1ST FREE MEM report; otherwise, Instant Assembler will respond "BAD" and ask for the origin again. Also, the origin must be low enough to allow the assembly to take place in the remainder of RAM; if it is not, Instant Assembler will reply "OUT OF MEM" and ask for the origin again.

When the assembly is finished, the total number of errors encountered will be reported. Also, the address that you entered in response to the "ORIGIN?" request will now be the origin of the source code; thus, if you list the program, the listing will correspond exactly to the assembled program.

RO (Reset Origin)

Use the RO command to define (or redefine) the origin of the source program. After the RO command is entered, you will be asked for an "ORIGIN?". Enter this in either decimal (five digits) or hexadecimal (four or fewer hex digits) -- see Appendix 5.

FR (Find References)

The FR command enables you to find (and edit, if you choose) all instructions in the source program that reference any specified symbol. Instant Assembler responds to the FR command with the query "FIND?". Answer this by entering any symbol that is in the source code. (If you enter a nonexistent symbol, Instant Assembler will merely repeat the "FIND?" question.) Instant Assembler will then display the first source line that references this symbol. The pause that follows (which is at the "line" level, as described under the ED command) gives you three options:

(a) BREAK to return to Instant Assembler command level. (b) DOWN ARROW or ENTER to find and display the next source line that references the specified symbol. (c) "C" to open the displayed line for editing -- exactly as with the ED command.

After the line has been edited (assuming the "C" option was exercised), Instant Assembler will find and display the next line that references the specified symbol. After the last line containing such a reference has been disposed of, Instant Assembler will ask for another symbol by repeating the "FIND?" question. Use the BREAK key to terminate the FR mode.

DI (Directory)

The DI command allows you to view a diskette directory without leaving Instant Assembler. This command functions under four operating systems: NEWDOS 80, DOSPLUS 3.4, Model III TRS-DOS, and Model III LDOS 5.1. As it comes to you, the DI command is set to work with Model III TRS-DOS; how to change it to work with another operating system is explained in Appendix 6.

After entering the DI command, you will be asked for a "DRIVE #?". Respond with "0", "1", "2", or "3", as appropriate. The directory will then be displayed. (With Model III TRS-DOS, only the names of the first 48 files on the disk will be shown.)

KL (Kill)

The KL command permits you to kill a file without leaving Instant Assembler. After typing the command, you will be asked for a "FILE NAME?". When this has been entered, Instant Assembler will reply "KILL. PROCEED (Y/N)?". Either type "Y" to kill the file or "N" to abort the KL command.

EX (EXit to DOS)

Use the EX command to exit safely to DOS.

MD (transfer to microMind)

Use the MD command to transfer control to MicroMind -- the debugging subsystem of Instant Assembler.

SECTION 2. EXAMPLE OF THE ASSEMBLER IN ACTION

Load and run Instant Assembler, and enter the CP command. Then compose the following source code. (Line numbers are furnished by Instant Assembler, of course.)

```
0001 ;HEX-TO-DECIMAL CONVERTER -- PART 1
0002 &BEGIN CALL 1C9H ;CLEAR SCREEN
0003 LD HL,3C14H
0004 LD (4020H),HL
0005 LD HL,TITLE
0006 CALL &VIDOT ;DISPLAY TITLE
0007 CALL &CARET
0008 INPLP CALL &CARET
0009 LD HL,HEXNM
0010 CALL &VIDOT ;DISPLAY PROMPT
0011 CALL &KBINP ;TAKE INPUT
0012 LD HL,&BUFFR
0013 CALL &CONVT ;CONVERT TO BINARY
0014 JR C,INPLP ;IF ENTRY IS BAD
0015 LD A,(4020H)
0016 AND 0C0H
0017 ADD A,11
0018 LD (4020H),A ;TAB 11 SPACES
0019 LD A,'<'
0020 CALL 33AH ;DISPLAY THE '<'
0021 LD A,20H
0022 CALL 33AH ;TAB 1 SPACE
0023 EX DE,HL ;BINARY NUMBER IN HL
0024 CALL 0A9AH ;SET TYPE FLAG
0025 XOR A
0026 CALL 1034H
0027 OR (HL)
0028 CALL 0FD9H ;CONVERT TO DECIMAL
0029 LD HL,4131H
0030 CALL &VIDOT ;DISPLAY DECIMAL NUMBER
0031 JR INPLP
0032 TITLE DEFM 'HEX-TO-DECIMAL CONVERTER'
0033 DEFB 0 ;MESSAGE TERMINATOR
0034 HEXNM DEFM 'HEX#? '
0035 DEFB 0
0036 ;
0037 ;PART 2 -- VIDEO OUTPUT AND CONVERSION ROUTINES
0038 &VIDOT LD A,(HL) ;NEXT CHARACTER
0039 OR A
0040 RET Z ;IF TERMINATOR
0041 CALL 33AH ;POST TO SCREEN
0042 INC HL
0043 JR &VIDOT
0044 &CARET LD A,0DH ;CARRIAGE RETURN
0045 JP 33AH
0046 &CONVT LD DE,0 ;INITIALIZE ACCUMULATOR
0047 NXTHX PUSH HL ;SAVE POINTER
0048 EX DE,HL
```

```

0049      ADD    HL,HL
0050      ADD    HL,HL
0051      ADD    HL,HL
0052      ADD    HL,HL
0053      EX     DE,HL      ;DE MULTIPLIED BY 16
0054      POP    HL        ;POINTER
0055      LD     A,(HL)     ;NEXT HEX DIGIT
0056      SUB    30H
0057      RET     C        ;BAD ONE
0058      CP     10
0059      JR     C,DIGIT   ;IF 0-9
0060      SUB    7
0061      RET     C        ;BAD ONE
0062      CP     16
0063      CCF
0064      RET     C        ;BAD ONE
0065 DIGIT  OR     E
0066      LD     E,A        ;ADD TO DE
0067      INC    HL
0068      DJNZ   NXTHX
0069      RET
0070 ;
0071 ;PART 3 -- KEYBOARD INPUT ROUTINE
0072 &KBINP LD     B,0      ;INITIALIZE CHAR COUNT
0073      LD     HL,&BUFFR
0074      LD     A,14        ;TO TURN CURSOR ON
0075 POST   CALL   33AH
0076      LD     A,B
0077      CP     4
0078      JR     Z,CRET      ;IF LIMIT IS REACHED
0079 NXTCH  CALL   49H      ;GET NEXT CHARACTER
0080      LD     (HL),A      ;PUT IN BUFFER
0081      CP     0DH
0082      JR     Z,CRET      ;IF ENTER KEY
0083      CP     31
0084      JP     Z,DOS       ;IF CLEAR KEY
0085      CP     8
0086      JR     Z,BKSPC    ;IF LEFT ARROW KEY
0087      INC    HL
0088      INC    B          ;INCREASE COUNT
0089      JR     POST
0090 BKSPC  LD     A,B
0091      OR     A
0092      JR     Z,NXTCH     ;IF NO CHARS ENTERED
0093      LD     A,(HL)
0094      DEC    HL
0095      DEC    B          ;BACK UP 1 CHAR
0096      JR     POST
0097 CRET   LD     A,15      ;TO TURN CURSOR OFF
0098      JP     33AH
0099 &BUFFR DEFS   4
0100 DOS   EQU    402DH

```

In entering the above program you may need to refer frequently to the procedures detailed under the CP command in subsection 1.1 of Section 1. When you have finished, you will have obtained a working knowledge of most of these procedures. (Did you use SHIFT-1, SHIFT-3, SHIFT-4 for DEFB, DEFM, and DEFS?) The on-line comments do not have to be aligned when you enter them; they will be aligned automatically in all listings. Also, if you have a Model III, you could enter the comments in lower case.

After the last line has been entered, press the BREAK key and type "LI". If you have done your work correctly, you should get the response, "ERR COUNT: 000". (If not, you may edit the error lines one at a time as they are displayed.) Then type "LE" to check that there are no external undefined symbols. Next, type "LS" and take a look at the symbol table. Note that the values of the symbols are low because an origin of 0 has been assumed for your program. You may use the RO command to change the origin to anything you wish. Now type "LC" and use the ENTER key to go through the entire source program 12 lines at a time, checking it carefully against the above listing. If you have a printer, turn it on, make it ready, and enter the PC command to obtain a printed listing of the program.

Now use the OS command to make a disk file of this program for later use. Give it the file name "HDCONV/SRC". Then, with the DM command, delete lines "&VIDOT-2" through "DOS". If you have done this correctly, only lines 1-35 (and the ORG and END lines) will remain, and there will now be eight "***UDS***" errors in the residual program. Since all these errors are references to external labels that will ultimately be resolved by Linking Loader, they are acceptable. Make a disk file of this segment of the program (with the OS command), giving it the file name "HDCNV1/SRC".

Next, use the IN command to read the HDCONV/SRC file. (You will have to override the source buffer protection feature to do this.) With the original program in the source buffer again, delete (with the DM command) lines 1 through "HEXNM+2" and lines "&KBINP-2" through "DOS", retaining Part 2 of the program. (Part 2 by itself should show no errors when listed.) Make a source file of this segment, using the title "HDCNV2/SRC". Finally, load the HDCONV/SRC file once more (with the IN command), delete lines 1 through "&KBINP-2", and make a source file of Part 3 (which also should have no errors), giving it the title "HDCNV3/SRC". Save these four disk files for later practice with MicroMind and Linking Loader.

By this time you have exercised many of the commands of Instant Assembler. To practice using the rest of the commands, read in the HDCONV/SRC file again. Enter the AM command, answer the "ORIGIN?" query with "9000", and press ENTER. Your program will be assembled into memory starting at 9000H, and Instant Assembler should report "ERR COUNT: 000". Transfer to MicroMind by typing "MD". Then enter the JP command, respond to the query "ADDRESS?" with "9000", and press ENTER. Your hex-to-decimal conversion program will now execute. Enter any hex number of up to four digits (pressing ENTER if the number of characters is less than four), and the number will be instantly converted to its decimal equivalent. When you tire of this, press the CLEAR key, and control will be transferred to DOS. From there you may reenter Instant Assembler by typing "IASTRF" and pressing ENTER. You will find the source buffer intact.

If you wish, you may make an object file of the hex-to-decimal converter. Use the OO command, and give it a file name of "HDCONV/CMD", an origin of 9000H (press ENTER for the default origin), and an entry point of 9000H (also by pressing ENTER in response to the query). Later, you can load and execute this file from DOS. Also, you may make an EDTASM source file with the OE command; save this for later review when you have EDTASM in your computer.

To see how the block move command operates, type "MB", and then move Part 3 of the program to just in front of Part 2. (FIRST LINE# = &KBINP-2, FINAL LINE# = DOS, and INSRT LINE# = HEXNM+2.) Next use the FR command to find all lines that reference the "&VIDOT" label. End this session with Instant Assembler by practicing inserting (IS), deleting (DL), and editing (ED).

(Since you may be interested in the inner workings of the hex-to-decimal converter, a few words are in order to clarify some of its more mysterious instructions. The program uses several ROM subroutines; otherwise, it would be much longer than it is. The ROM subroutine at 1C9H clears the screen. The one at 33AH displays a character at the cursor position and updates the cursor. The subroutines at 0A9AH, 1034H, and OFD9H act to convert a 16-bit binary number to a string of decimal digits. The subroutine at 49H scans the keyboard and decodes the input characters. Locations 4020H-4021H contain the address of the video memory cell in which the cursor resides. Any remaining mysteries could be solved by using MicroMind to step through the program.)

SECTION 3. INSIDE INSTANT ASSEMBLER

This section is a collection of tidbits and hints -- information about your assembler that you will eventually want to have.

(1) Whenever you want to exit to Instant Assembler's command level, use the BREAK key. The only time that this won't work is when you are editing a source line, in which case BREAK acts like ENTER.

(2) The line number of any instruction -- instead of being fixed as it is in EDTASM -- is determined by the relative position of the instruction in the source buffer, which may change as a result of insertion, deletion, or block movement. This implicit line numbering has several advantages over fixed line numbering. For one thing, it allows continuous insertion of new lines without periodic interruptions for renumbering. Also, the block movement command is much easier to implement with implicit line numbering. And -- not least -- implicit line numbering saves a great deal of space in the storage of the source code. Of course, there are some disadvantages, too. One of these shows up when you want to delete two or more nonadjacent lines of code. For example, suppose that you wish to delete lines 69 and 85; if you first delete line 69, you will find that the other line is now number 84. One way to handle this problem is to delete from the top down. (In the example, delete line 85 first.) Another way is to address each line by its label (plus offset); in fact, this ability to address a line by its label (plus offset) frees you from dependence upon absolute line numbers. As a last resort, you can always find the up-to-date line number of any instruction by using the LL command.

(3) There are limits to the size of a source module that can be constructed. The first limit is imposed by memory size. If you run out of memory while composing a program, Instant Assembler will report "OUT OF MEM" and exit to command level. Another limitation is that the total number of symbols may not exceed 1024. If a newly entered symbol would exceed this limit, composition is halted with a "SYMB OVF" (symbol overflow) message. The final constraint is that your program's length should not exceed 32768 bytes (of object code). The only possible way that this limit could be exceeded before you run out of memory is for you to reserve a great amount of storage (with DEFS's) in your program. Instant Assembler provides no protection against this most unlikely overflow; you are responsible for seeing that it doesn't happen.

(4) Deletions from Instant Assembler source code do not result in deletions from the symbol table of that source code. If a module goes through many revisions, it may collect a number of dead symbols in its table. Since these dead symbols still apply to the 1024 limit of (3) above, it might be desirable at some time to purge the symbol table. This can be done by merging (with the MG command) the source module into an empty source buffer.

(5) When a listing to the screen pauses after 12 lines, you are still in listing mode. If you wish to edit one of the displayed lines, press BREAK (or almost any other key) before typing the ED command.

(6) If you want to move a line that has a label, either use the MB command, or else delete the line before reinserting it. An attempt to insert the line before it is deleted will result in a "DBLY DFND LABEL" error.

(7) For the most part, numeric constants are listed in the same form in which you enter them. However, there are a few exceptions. All index register offsets are listed in decimal. The operand of any RST instruction (if greater than 8) is listed in hexadecimal. All 16-bit hex values are listed with four or five hex digits; thus, CALL 60H is listed as CALL 0060H. (This quirk is at the request of Bryan Mumford.) 8-bit hex constants are listed without unnecessary leading zeroes, however.

(8) Origins and entry addresses entered in hexadecimal do not require a terminating H or a leading zero. This feature is for your convenience.

(9) When the source buffer is empty, the message "NO CODE" will be displayed in response to any command that would operate on the source code.

(10) External labels are those that commence with an ampersand. You may use them whenever you feel like it; all your labels may be external if you choose. However, you need to use them only if the program module that you are composing will be loaded together with other modules that reference it. Then, every instruction and storage location in this module that will be referenced by another module must be given an external label. Later, Linking Loader will be able to assemble and link all the modules. (Linking Loader does not check for doubly defined external labels, so you must be careful that you use each external label in only one module. Any nonexternal label may be used in as many modules as you please.)

(11) An instruction like LD HL,STORE+512 is not accepted by Instant Assembler because of the size of the offset. To construct an equivalent instruction, simply use another label closer to the target address. (A DEFS instruction ahead of the new label may be necessary for correct positioning.)

(12) An instruction like LD BC,END-BEG+1 presents a harder problem, and may require the expenditure of a few additional bytes of code. The following coding will always suffice, and may be shortened (by deleting the PUSH and POP) if the HL register pair is free at the time.

```
PUSH HL
LD HL,END+1
LD BC,BEG
OR A
SBC HL,BC
LD B,H
LD C,L
POP HL
```

(13) Although Instant Assembler does not permit the use of symbols to represent 8-bit values, there is usually an easy way around this limitation, too. A typical example of the use of an 8-bit symbol is the following:

```
OUT PORT1,A
```

with the symbol PORT1 defined by means of an EQU. (The purpose of this is to make it easy to change all port numbers by changing one EQU.) The following Instant Assembler-compatible code will accomplish the same objective:

```
DEFB OD3H ;"OUT" INSTRUCTION
DEFW PORT1 ;PORT NUMBER
```

The 16-bit value for PORT1 (established by an EQU) will have a high-order byte of zero, which is a NOP to the Z-80.

(14) Instant Assembler has been carefully designed to make it difficult for you to wipe out the source buffer inadvertently. If you want to do this deliberately, use the CP command, respond "Y" to the "CODE ERASURE. PROCEED (Y/N)?" query, then press BREAK.

(15) If your printer has trouble working with Instant Assembler, or if you would like to change the format of printed listings, make a careful study of Appendix 6. Then, try changing some of the print parameters.

(16) When Instant Assembler prompts you for information (FILE NAME?, ORIGIN?, etc.), it does not permit you to enter more characters than the maximum number that any correct answer could use. If you enter this maximum number of characters, Instant Assembler will immediately analyze your response; you do not have to press the ENTER key in this case. Exception: Your answer to the "TITLE?" query must be completed by pressing ENTER.

(17) The upper and lower halves of the IX and IY registers can be independently addressed through a number of undocumented Z-80 instructions. These half-registers are given the names IXH (IX high), IXL (IX low), IYH, and IYL. For example, CP IXH is a two-byte instruction that compares the A register with the upper eight bits of the IX register. The instruction forms in Appendix 1 that contain the symbol "x" as an operand correspond to the undocumented instructions. Instant Assembler will recognize and assemble any of these forms. These instructions provide you with four additional 8-bit registers, with an overhead charge of one extra byte of object code per instruction.

(18) Instant Assembler source code is closer to object code than it is to text. The actual format is as follows: Each instruction is preceded by a bit-encoded header byte that contains information about the type and length of the instruction, whether the instruction has a label, whether it references a symbol, and whether an extension byte is needed to provide further information. This header byte is followed by the object code for the instruction, except that a symbolic address is represented by the number of the symbol (that is, its position in the symbol table) rather than the value of the symbol. If an extension byte is needed, it follows the object code. An extension byte may contain an extension of the symbol offset, format information for listing the instruction, and a flag to indicate an on-line comment. Any on-line comment then follows the extension byte (headed by a byte giving its length). In addition to the code structure just outlined, Instant Assembler source also contains a symbol table. Each symbol is kept in 10 consecutive bytes of the table; six of these bytes hold the actual characters of the symbol, two bytes contain the value of the symbol, and two bytes serve as a linked list pointer to the next symbol in alphabetic order. The "value" of a symbol -- other than an EQUated symbol -- is its offset from the origin of the source code module.

(19) If at any time you find yourself back in DOS and want to return to Instant Assembler without destroying the source buffer or resetting the memory protection (in 5AFEH-5AFFH), merely type "IASTRF" and press ENTER. IASTRF employs the nondestructive reentry to Instant Assembler at 5B03H, an address that you may use from any other program that has a transfer capability.

PART II. THE DEBUGGER

Instant Assembler's debugger is named MicroMind, and it is supplied in two forms. The integral MicroMind is contained within the DSKIAS/CMD package and is reached via the MD command from the assembler subsystem. The stand-alone MicroMind is the MICROM/CMD file on your diskette. The two versions are identical except for a few commands and the relocating feature of stand-alone MicroMind. The command differences are treated in Sections 4.4 and 4.5, and the procedure for relocating stand-alone MicroMind is explained in Section 6, item (1).

SECTION 4. MICROMIND COMMANDS

MicroMind has 21 two-letter commands, which will be fully explained in this section. (They are also summarized in Appendix 7.) As in the assembler subsystem, response to your command follows immediately upon typing the second letter of the command. The command prompt in MicroMind is "**". The BREAK key can be used at almost any time to return to command level.

4.1. Stepping, Breakpointing, and Executing

SP (Step)

This command puts MicroMind into the step mode, which allows you to step through a machine language program one instruction at a time. This mode with its register displays provides a nearly infallible tool for debugging. After you have entered the SP command, you will be asked for a "FIRST ADDRESS?", which is the address where you will begin to execute the machine language program step-by-step. This address may be entered in either decimal or hexadecimal. The rule for the entry of all addresses is: If five digits are entered, the address is in decimal; if fewer than five digits are entered, the address is in hexadecimal. (This rule is also explained in Appendix 5.)

Once you are in step mode, the registers will be displayed at the top right of the screen, and you may fetch and execute instructions merely by pressing ENTER. As each instruction is fetched, it is displayed in two lines at the top left of the screen; the first display line shows the memory address and the hex bytes of the instruction, while the second line shows the Z-80 mnemonic form of the instruction.

ENTER actually half-steps through the target program. You see the FETCH and EXECUTE cycles as separate half-steps, each activated by pressing ENTER. After the EXECUTE cycle, another register display at the bottom right of the screen shows the contents of the registers after execution of the instruction; the BEFORE register display remains in the upper right corner of the screen, so that the effect upon the registers of the instruction just executed can be clearly seen. When the next instruction is fetched, the BEFORE display will change to contain the information of the AFTER display from the previous execution, and the AFTER display will be erased.

The register displays are largely self-explanatory. Each double register is presented with a designator followed by the (hex) contents of the register. The most important flags are also displayed as separate bits; their designators are CY for Carry, Z for Zero, S for Sign, and PV for Parity/oVerflow. Following the BC, DE, HL, IX, and IY displays are single hex bytes in parentheses; these represent the contents of the memory locations pointed to by these 16-bit registers. For example, if you see DE: 48FB (E6) in the display, you know that the DE register pair contains 48FBH and that memory location 48FBH contains 0E6H. The two-byte hex number in parentheses following the SP (stack pointer) display is the number (or address) on the top of the stack. This number is presented in the natural form of high-order byte first, even though the high-order byte is in the memory cell whose address is one more than the contents of the SP register. The PC display shows the contents of the program counter, which points (in the BEFORE display) to the memory location from which the instruction has been fetched, or (in the AFTER display) to the memory location from which the next instruction is to be fetched.

In the SP mode you may fast-step through several (up to 99) instructions by typing a two-digit number in response to the "*" prompt. For example, entering "03" will cause three complete instructions to be executed in rapid succession. The register displays show the register contents before the first instruction is executed and after the last instruction has been executed. (This feature of MicroMind is especially helpful in working through a short loop that has to be executed many times.)

NOTES: (1) Once you are in SP mode, you will not permanently exit from it except with a JP (JumP), CL (Call), or (in integral MicroMind) IA (transfer to Instant Assembler) command. However, you can reinitialize the mode with another SP command. (2) When you are in SP mode and the "*" is displayed, pressing ENTER will always cause the next half-step (FETCH or EXECUTE) to be carried out. This is true even if you have made extensive use of other commands since your last step. To clear the left side of the screen so that the FETCH or EXECUTE cycle can be clearly observed, merely press BREAK. (3) Not all hex numbers can be decoded as legitimate Z-80 instructions. In the unlikely event that MicroMind encounters such an indecipherable combination in the instruction stream while stepping, it will treat each byte as a NOP for execution (and display no mnemonic for that byte) until it arrives at the next recognizable instruction. Further discussion of this point will be found under the DS command in Section 4.3.

XC (eXeCute)

XC is operative only in step mode, and only if the last instruction fetched is a CALL, conditional CALL or RST (restart). Its effect is to execute the called subroutine as a whole, without stepping. It is useful when the target subroutine has already been debugged. The register displays show the contents of the registers before the subroutine is called and after it has been executed. If the XC command is entered when it is inapplicable, it is simply ignored, and another "*" prompt is issued. If the fetched instruction is a conditional call, and if the condition is not met, the XC command merely has the effect of stepping through the conditional call instruction without executing the subroutine.

BD (Blank Display)

This command is operative only in step mode. Its effect is to clear the screen (except for less than half a line in the extreme lower left corner), to transfer an abbreviated display of the fetched instruction to the lower left corner of the screen, and to permit continued stepping with the target program in control of the screen (except for the half-line in the lower left corner). Pressing ENTER now causes execution of the fetched instruction and fetching of the next instruction in sequence (full-stepping rather than the half-stepping of SP mode). No register displays are available in BD mode; the target program can be traced, but a detailed examination of its workings is no longer possible.

To return from BD mode to regular SP mode, merely press the BREAK key; there will be no loss of place or of continuity in stepping through the target program.

The BD mode has been designed to allow you to see a target program print on the video screen. Since many instructions are normally required to post even one character to the screen, single-stepping through a video display routine can be distressingly slow. Therefore, MicroMind provides three sub-commands in BD mode to speed up the action:

(1) Depressing the R (for Run) key while in BD mode causes stepping to occur at about 140 instructions per second. Releasing the R key terminates the fast-stepping and returns you to the single-stepping (via the ENTER key) of BD mode. (2) The S (for Seek subroutine) key has the fast-stepping effect of the R key, except that fast-stepping is terminated whenever a CALL, conditional CALL, or RST is encountered in the instruction stream. Often keyboard input requests are mixed in with video output, and stepping (even fast-stepping) through an input routine can be an aggravation. Use of the S key in BD mode allows you to fast-step until a subroutine call is reached, then pause long enough to use the X key. (3) The X (for eXecute) key permits execution of the subroutine as a whole. The X key is to BD mode what the XC command is to SP mode.

RN (RuN)

RN is also operative only in SP mode; this command enables fast-stepping (with a blank screen) to any designated terminal address. After you have entered the RN command, you will be asked for a "FINAL ADDRESS?". This is the address of the last instruction to be fetched. (The run will be ended when this instruction is fetched but not executed. The run starts, of course, from the instruction that you have reached in SP mode.) The FINAL ADDRESS should be entered in accordance with the usual rule for entering addresses (Appendix 5).

The RN command permits you to execute rapidly a portion of the target program without leaving or reinitializing the SP mode, and with the assurance that your designated terminal point (which may even be in ROM) will be honored. When the final address has been reached in RN mode, the fast-stepping will stop, and you will be in BD mode.

If you wish to exit from RN mode before the terminal address is reached, press the BREAK key. You will then be in BD mode, from which you may continue with single-stepping, or use the R, S, and X keys, or return to SP mode by pressing BREAK once more. Note, however, that if you now use the R key, it will return you to RN mode (running to the designated final address); that is, releasing the R key now will not terminate the (still unsatisfied) run -- only the BREAK key can do that in this instance.

BK (Breakpoint)

The BK command allows you to set a breakpoint in RAM. After "* BK", you will be asked for an "ADDRESS?". This should be the address of the first byte of some instruction in the target program. (MicroMind will not accept any address lower than 4000H.) MicroMind will then replace three bytes of the machine language program with a jump to a MicroMind entry point and will confirm the breakpoint with the message "BREAK AT (address)". (The three replaced bytes are saved for later restoration.) If the BK command is entered when a breakpoint is already in effect, the same message will appear (with the old breakpoint address) in rejecting the command.

Upon return from a breakpoint, an AFTER register display will appear in the lower right corner of the screen, showing the contents of the registers at the completion of the program segment terminated by the breakpoint.

RB (Restore Breakpoint)

RB undoes the effect of BK. It may be used if you change your mind about the breakpoint that you have set, or it may be used after returning from a breakpoint. Since only one breakpoint can be in effect at any one time, RB might be used to restore an old breakpoint so that a new one may be set. When the RB command is entered, the confirmation "BREAK AT (address) RESTORED" will appear, assuming that a breakpoint was actually in existence; if there was no breakpoint, the legend "NO BREAK" will be displayed.

SB (Step from Breakpoint)

SB combines the effects of RB and SP, with the latter commencing at the address of the breakpoint. That is, the breakpoint will be restored, and the instruction at the address of the breakpoint will be the first one fetched for step-wise execution. SB is useful after a return from a breakpoint, at which time the registers will be in exactly the condition in which the program segment just executed has left them. If the SB command is entered when no breakpoint is in effect, the legend "NO BREAK" will appear.

JP (Jump)

The JP command allows you to transfer control to any point in memory, including ROM. After "* JP", you will be asked for an "ADDRESS?". (To change your mind at this point, use the BREAK key to cancel the JP command.) When this address has been entered, the registers will be loaded with the values shown in your last register display (if you were in SP mode), and the jump will be taken to the specified address. JP is useful in conjunction with a breakpoint; after setting a breakpoint with BK, use JP to execute part of a machine language program and then return to MicroMind.

It is obvious that caution must be exercised in the use of JP, since control is taken out of the hands of MicroMind. In particular, if the jump address is five digits (decimal), be certain that you have entered it correctly before typing the last digit, for the fifth digit automatically triggers the jump.

CL (Call)

The CL command allows independent execution of any closed subroutine in ROM or RAM. After entering the CL command, you will be asked for an "ADDRESS?". Respond with the entry point address of the subroutine to be executed. As soon as the address entry is completed, the screen is cleared, the registers are loaded with the values shown in your last full register display (if you were in SP mode), and control is transferred to the target subroutine. (If you wish to adjust the contents of the registers before calling the subroutine, use the RG command, which is explained in the next section.) The return will be to MicroMind. You may inspect the contents of the registers after execution of the subroutine by using the SP command (with any FIRST ADDRESS) to call up a full register display.

4.2. Register and Memory Display

RG (ReGister display and change)

The RG command allows you to inspect and change the contents of a target program's registers. Following "* RG", you will be asked to name a register by the query "REG?". You may answer this question with "A", "CY" (for Carry), "Z" (for Zero), "S" (for Sign), "PV" (for Parity/oVerflow), "BC", "DE", "HL", "IX", "IY", or "SP" (for Stack Pointer). A two-letter name automatically triggers the display of the contents of the named register; if your request is for the A, Z, or S register, you must press ENTER to trigger the display. Note that the flags can be inspected and changed as conveniently as any of the other registers.

When the register's name has been entered, its contents will be displayed to the right of the name, followed by a LESS THAN prompt. To change the contents of the register, enter a new byte (in hex) for the A register, either "0" or "1" for a flag, or a new number (in address format -- either five decimal digits, or four or fewer hex digits) for a double register. (If you enter a new byte with only one hex digit for the A register, you will have to press ENTER to complete the entry.)

If you were in SP mode before using the RG command, and if you were between FETCH and EXECUTE cycles, then all register changes will be immediately reflected in the BEFORE register display. Such changes will not be shown, however, after an EXECUTE cycle -- they will appear after the next FETCH cycle.

If you don't wish to change the contents of a displayed register, just press ENTER. Completing an entry on one line (either by entering a change for the register or by pressing ENTER) will cause a new "REG?" query to appear on the next line. To exit RG mode, use the BREAK key, either in response to the LESS THAN symbol that prompts the register change, or in response to the "REG?" query. You will be returned to SP mode if you were there before using RG, and you may then continue stepping through a program.

The RG command is useful in debugging for the following reason (among others): Often you will discover an error that adversely affects register contents. Rather than having to abort the debugging procedure to make a change in the program, you may make a note of the discovered error, use RG to set the registers right, and then continue stepping.

MM (MeMoRY display and change)

MM allows you to inspect and change the contents of memory locations. (ROM may be inspected, too, but not changed.) After "*** MM**", you will be asked for a "FIRST ADDRESS?", which again may be in either decimal or hex. After a memory cell is displayed (one hex byte following the hex address of the cell), a LESS THAN prompt will appear, and you may change the contents of the cell by typing your new byte (which must be in hex). (If you type a single character here, the transaction must be completed by pressing ENTER. A two-character entry automatically triggers the change.) When the change has been entered, the display advances to the next memory cell. If you press ENTER without entering a change, the display also advances.

In integral MicroMind, if you press the DOWN ARROW (or the UP ARROW) key without entering a change, the display will be advanced (or backed up) one memory location. (These features are not implemented in stand-alone MicroMind.)

If you want to back up, or to advance, one or more addresses, use the MINUS (-) key, or the PLUS (+) key, respectively, followed by the number of addresses that you wish to back up or advance. (These two characters are typed in lieu of a hex byte for changing the contents of the memory cell.) Following the MINUS or PLUS, digits 0-9 will have their natural effect, while letters A, B, C, ..., Z will back up or advance the address by 10, 11, 12, ..., 35 locations (respectively).

To exit from MM mode, use the BREAK key. If you were previously in SP mode, you will be returned precisely to where you were before you used the MM command -- except, of course, that some memory cells may have been changed. (If you were between FETCH and EXECUTE cycles, the instruction fetched will remain the same even if you changed it in memory during the MM operation.) Memory changes will be immediately reflected in the BEFORE register display if (and only if) you are between FETCH and EXECUTE. (Remember that the register display shows the contents of those memory locations that are pointed to by the double registers BC, DE, HL, IX, IY, as well as the two-byte number on the top of the stack.)

AS (AScii display)

The AS command allows you to decode blocks of memory as ASCII characters. After "*** AS**", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the block of memory that you wish to examine. MicroMind will then display five lines of 10 characters each and pause, awaiting your next directive. Pressing either the DOWN ARROW or the ENTER key will cause the next five lines of 10 characters to be displayed, while pressing the UP ARROW key will cause the display to back up 50 characters. Press BREAK to end the AS mode.

Here's an example of the use of the AS command: With MicroMind running, enter the AS command and give it an address of 1650. You will then be reading the start of the BASIC command table in ROM. Press DOWN ARROW repeatedly to scan this table. Note the small graphics block at the upper left corner of the initial letter of each command. This block indicates that the character in memory has bit 7 set. A graphics block at the left middle of a character indicates that it is actually a lower case character. A two-wide graphics block where a character should be indicates that the number in memory cannot be deciphered as an ASCII character. And, finally, a three-tall graphics block where a character should be indicates a carriage return.

P1 (Page 1)

The P1 command is used to display a small block of memory. After "* P1", you will be asked for a "FIRST ADDRESS?". When you have responded, 19 consecutive memory cells starting at this address will be displayed at the next-to-bottom line of the screen. (Page 2 will simultaneously be displayed at the bottom line.) The two-byte hex number at the extreme left of the P1 display line is the address of the first memory cell of the display.

When MicroMind has posted the P1 display, it will wait until you press another key. Pressing either the DOWN ARROW or ENTER key will advance the starting address of the P1 display by 10H, while pressing the UP ARROW key will back up the display by 10H. Either of these actions may be repeated for as long as you desire; thus, you may quickly scan a large section of memory. Press the BREAK key to return to the facilities of MicroMind.

When MicroMind is first activated, Page 1 will not be displayed until either the P1, P2, or SP command is invoked, or memory is changed with the MM command. When this occurs, both Page 1 and Page 2 will appear on the screen, and their displays will remain in evidence until you exit from MicroMind. A return from a breakpoint will reactivate the page displays, but other reentries will not.

P2 (Page 2)

P2 is exactly like P1, except that the display is on the bottom line of the screen. P1 and P2 may be set independently, so that you can keep an eye on two different regions of memory, which can be extremely helpful in debugging.

4.3. UTILITIES

FN (Find Number)

The FN command allows you to find all occurrences of either a one- or two-byte number in a block of memory. (This can be especially useful for finding all program references to a certain address.) After "* FN", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the block of memory to be searched. When this has been accepted, MicroMind will ask you for a "FINAL ADDRESS?". Respond with the last address of the block of memory to be searched. Then the question "FIND?" will appear. Answer this with the number that you wish to locate in the specified memory block. (Entry of this number follows the usual rule for entry of addresses given in Appendix 5.) If you enter three, four, or five characters here, MicroMind will search for a two-byte number (or address); if you enter only one or two characters, MicroMind will search for a one-byte (hex) number. (In the latter case, expect a lot of matches unless the memory block is small.) The addresses of all occurrences of your search number in the specified memory block will then be displayed, four to a line. (For a two-byte search number, the address shown will be that of the low-order byte.)

After MicroMind has reported all addresses corresponding to your search number, it will post another "FIND?" query. By entering another number here, you may continue searching the same block of memory. To exit from FN, press BREAK.

DS (DiSassemble)

The DS command enables you to see instructions in memory (including ROM) in their Z-80 mnemonics. After "*** DS**", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the program that you wish to disassemble. The instruction at that address will be disassembled and displayed in the format explained under the SP command. MicroMind will then pause, awaiting your next directive. Pressing either the DOWN ARROW or the ENTER key here will cause the next instruction in memory to be displayed, while pressing the UP ARROW key will result in the display of the instruction at memory address 10H lower than that of the next instruction. Either of these actions may be repeated for as long as you wish. Press the BREAK key to end the DS mode.

NOTE: Not all hex numbers can be decoded as legitimate Z-80 instructions. For example, the byte ODDH by itself is meaningless -- it requires at least one following byte to give it meaning. And not all following bytes are legal; DD01 is not the beginning of any valid Z-80 instruction. When the disassembler encounters such a combination, it reports only the first byte (with no mnemonic) on a line, and then proceeds (as you press DOWN ARROW or ENTER) one byte at a time until it finds a combination that it can decode. In disassembling data, this type of ambiguity can easily arise. When you have worked through the data, the MicroMind disassembler will quickly get back into "sync", though an instruction or two following the data may be misreported. In stepping through a program (with SP), the same impasse is possible, though far less likely than in random disassembly. In this unlikely event, MicroMind treats the unidentifiable code byte-by-byte -- turning each byte into a NOP for execution -- until it can get back into "sync".

HD (Hex-to-Decimal conversion)

After "*** HD**", you will be asked for a "HEX#?". Enter any number of not more than four (hex) digits, completing the entry by pressing ENTER if fewer than four digits are typed. MicroMind will respond with the decimal equivalent of that hex number. It will then ask for another "HEX#?". Terminate this mode by pressing BREAK. (The hex-to-decimal converter of Section 2 is similar to this MicroMind routine.)

DH (Decimal-to-Hex conversion)

This conversion works like the HD above, except that now you will enter a decimal number of up to five digits, and the response will be its hex equivalent. (In either type of conversion, if MicroMind can't decipher your input, it will simply ask for it again.)

4.4. Symbolic Disassembly and Transfer (Integral MicroMind Only)

SD (Symbolic Disassembly)

When you assemble-to-memory a program in Instant Assembler's source buffer and then use (integral) MicroMind either to step through it or to disassemble it, you might want to see the disassembled instructions with labels, symbols, character constants, etc. -- exactly as they would appear in a listing of the source code. To do so, enter the SD command before beginning to step or disassemble.

During symbolic disassembly (or stepping), if an instruction is encountered that does not agree with its counterpart in the source code, the actual instruction in memory is the one that will be disassembled and displayed. (This could happen because of a memory change, for instance.)

AD (Absolute Disassembly)

The AD command is used to switch back to normal disassembly after you have had enough of symbolic disassembly. (You can switch back and forth between the two as often as you like.) Whenever you enter MicroMind with the MD command (from the assembler), the absolute disassembly mode will be in effect.

IA (transfer to Instant Assembler)

Use the IA command to transfer control to the assembler subsystem.

4.5. Tape and Printer Commands (Stand-Alone MicroMind Only)

TP (TaPe)

The TP command allows you to record a 500 baud machine language program (in SYSTEM format) on cassette tape. You may use it to make an object tape of any machine language program that is in memory. After you have entered the TP command, MicroMind will request a "FIRST ADDRESS?". This is the lowest memory address of the program that you wish to record, and it may be entered in either decimal or hex. Next, you will be asked for a "FINAL ADDRESS?". Enter the last memory address of the program to be recorded. You will then be asked for an "ENTRY ADDRESS?", which is the address at which execution of the machine language program is to start when later it is loaded with the SYSTEM command. Finally, MicroMind will request a "TITLE?"; type in any name of six or fewer characters. (First character must be a letter, subsequent characters either letters or digits.) Have the tape correctly positioned in the recorder, with the RECORD and PLAY keys depressed; as soon as you type the title and press ENTER, the recording will commence.

VF (VeriFy)

The VF command allows you to verify a 500 baud machine language tape that you have just recorded with the TP command. Rewind the tape to the beginning of the recorded segment and type "VF" in response to the "*" prompt. Press the PLAY key on the recorder, and MicroMind will try to verify the recording. If anything is wrong, "BAD" will be displayed, and you may re-record the program

and try again to verify it. If the recording is all right, "GOOD" will be displayed when the verification is complete.

DP (Disassemble and Print)

The DP command allows you to print a disassembled listing of any segment of Z-80 code in ROM or RAM. After entering the DP command, you will be asked for a "FIRST ADDRESS?" and a "FINAL ADDRESS?". (These are, of course, the first and last addresses of the block of code that you wish to disassemble.) Enter these addresses in either decimal or hex. Have your line printer turned on and ready. When the FINAL ADDRESS has been entered, printing will commence and will continue until this address has been reached or exceeded, or until you depress the BREAK key to stop the operation.

SECTION 5. EXAMPLE OF MICROMIND IN ACTION

The goal of this section is to encourage you to become familiar with MicroMind by practicing its operations on the hex-to-decimal conversion program that you composed with Instant Assembler in Section 2. To that end, a program of action is described, but detailed instructions for carrying out all of the steps are not given; refer to the explanations of Section 4 for any required additional help.

Load and run Instant Assembler. Use the IN command to load the HDCONV/SRC file that you recorded with the directions of Section 2. Assemble-to-memory (with the AM command), giving the program an origin of 9000 (which is hex). If you have a printer, make a listing of the assembled program with the PC command; this will help in following the steps outlined below. Now transfer to MicroMind; the command for this is "MD". Enter the SP command and an address of 9000. You are ready to commence stepping through the hex-to-decimal converter.

For a first run, a blank screen will allow you to follow the highlights of the action. Enter the BD command, and the screen will be cleared except for the CALL 1C9H instruction displayed at the lower left corner. Press the X key to execute this subroutine. Then press ENTER three times to step through the instructions at 9003, 9006, and 9009. Type "X" to execute the video output routine called at 900C, and the program title will appear on the screen. Press the X key twice more to execute the next two subroutines, which merely move the screen display line down. Then step through the instruction at 9015, and type "X" to execute the next subroutine, which will display the "HEX#?" prompt. Press the X key again, and the cursor will appear; you are now in the input routine. Enter a hex number of four digits, and you will be back in MicroMind, ready to step through the instruction at 901E. Continue stepping -- using the X key to execute each CALL that you encounter -- until the decimal equivalent of your hex entry has appeared, and the instruction at 9012 has been fetched again. (Do not be alarmed if the hex number appears to be converted incorrectly. MicroMind uses the ROM conversion facility for its own purposes between the actual hex conversion and its display.)

Press the X key to execute the subroutine called at 9012, and step through the instruction at 9015. Now, instead of executing the video output routine called at 9018, press ENTER to step through the CALL, then press the S key to fast-step until the instruction at 9071 (CALL 33AH) is reached. Use the X key to execute this subroutine, then fast-step (with the S key) until you return to 9071. Continue in this fashion, using the X key each time you reach the instruction at 9071. You should see the "HEX#?" query take form on the screen one character at a time. When this has occurred, press BREAK to return to SP mode.

Now enter the SP command again, with an address of 9003. Enter the RN command, giving a FINAL ADDRESS of 901B. MicroMind will then fast-step (with a blank screen) until the CALL to the keyboard input routine is fetched, at which point it will terminate the stepping. During this run you should see the program title and the "HEX#?" prompt appear slowly on the screen. Use the BREAK key to return to SP mode.

Before stepping again, enter the SD command so that you can see the labels and symbols of the fetched instructions. Then enter the SP command once more, with an address of 9003. Step through the program with full register displays; use the ENTER key to fetch each instruction, and also to execute it -- unless it is a CALL, in which case use the XC command to execute the subroutine as a whole. Note that the screen displays of the hex-to-decimal converter will now flash and be gone; MicroMind needs the screen for its own displays. (Don't be

alarmed if there is a bit of residue from the converter display, or if the two displays are sometimes intermixed.) When you execute the keyboard input routine (called by the instruction at 901B), however, you will be able to see your input until the fourth digit is entered. That is because MicroMind is suspended while the input subroutine is executing. In stepping with full register displays, proceed very deliberately, observing how the registers are affected by each instruction.

It is illuminating to step through the program again with a slight modification along the way. Enter the SP command and an address of 9000. Execute (XC) the instruction at 9000, and step through the instruction at 9003. Now use the RG command to change the contents of the HL register pair to 3E14. Exit from RG mode with the BREAK key, enter the BD command, and continue stepping (using the X key to execute all CALLS.) The effect of the register change is to move the initial display half way down the screen. When you have observed this, press the BREAK key.

As a final exercise, use the BK command to set a breakpoint at 90AE. (This is inside the keyboard input routine.) Then use the JP command to transfer control to 9000. The hex-to-decimal converter will now execute without external support. However, as soon as you enter a first character in response to the "HEX#?" prompt, the breakpoint will take effect, and MicroMind will be in control again. Now enter the SB command, and you can step through the processing of this input character. If you want to see how different input characters are processed, you can reinitialize the SP mode with an address of 90AE, then use the RG command to enter a character directly into the A register. (You might put zero into the BC register at the same time to avoid bumping up against the field limit of four characters.) Step from this point, and you will be able to observe the handling of the character. This last procedure shows how easy it is to back up -- or advance -- the stepping process by simply reinitializing the SP mode; frequently this needs to be accompanied by use of the RG command to set the registers right for the new starting point. Be warned, however, that reinitializing SP mode also reinitializes the stack pointer, so that any data or return address on the stack will be temporarily lost; if you wish, you can use the RG command to reset the stack pointer to its previous value.

It should be evident that, in the process of single-stepping through a program, you will almost surely discover any error that exists. If the program resides in Instant Assembler's source buffer, it is a simple matter to transfer to Instant Assembler (with "IA"), correct the error in the source code, assemble-to-memory again, and return to MicroMind (with "MD") to continue debugging.

SECTION 6. INSIDE MICROMIND

Herein is a collection of tips on the use of MicroMind.

(1) Stand-alone MicroMind is relocatable. When you run this program from DOS, the title "RELOCATOR" is displayed, and you are asked for a "STARTING ADDRESS?" for the relocation. If you are satisfied with the location of MicroMind as it is loaded from the disk, press the BREAK key; you will then be in MicroMind. If you want to relocate, enter a suitable starting address. After the relocation (which takes about four seconds), the relocator will report the FIRST ADDRESS, the FINAL ADDRESS, and the ENTRY ADDRESS for the relocated program; these addresses may be used to make a recording of the relocated program. (This recording can be made with the 00 command of one of the Linking Loaders. See Section 7 for directions on how to do it.) After these addresses have been displayed, control is passed to (old) MicroMind; you can use the JP instruction to transfer to the relocated MicroMind. To help you judge whether to relocate MicroMind, as well as what the starting address should be, the following information is supplied: Stand-alone MicroMind is about 1100 (hex) bytes long. As it comes off the disk, it occupies memory from about AF00 (hex) to exactly BFFF (hex). The relocator will not permit an overlapping relocation, a relocation below 5200 (hex), or one above FFFF (hex).

(2) A leading zero (or zeroes) is required when entering a decimal address of less than five digits; otherwise, MicroMind will treat the entry as a hexadecimal address. Also, when using the FN command to search for a two-byte hex number whose value is less than 100H, a leading zero will be necessary to tell MicroMind that it is a two-byte number (rather than a single byte) that you are searching for; if your two-byte search number is less than 10H, two leading zeroes will be required. Note that the "H" prefix is never required (or even allowed) for hexadecimal numbers.

(3) Each MicroMind query has a strict limit on the number of characters that can be entered in response. In all cases except for the "TITLE?" query, when this limit has been reached, the entry is complete, and the subsequent action is immediately triggered.

(4) The BREAK key allows you to escape from any mode except SP mode. In SP mode, BREAK merely clears the left side of the screen.

(5) If MicroMind cannot recognize your response to a prompt or query, it repeats that prompt or query.

(6) MicroMind accepts no space in its input, because none is needed in any of the formal responses to its queries.

(7) Use of other commands will not confuse the stepping process. MicroMind will never forget where it is in the stepping process until you do a JP, CL, IA, SB, or another SP, and exiting from any of the other commands will automatically return you to where you were in SP mode.

(8) To view all the registers at once, enter the SP command to call up a full register display. It isn't necessary to do any stepping to use this feature.

(9) You can't directly inspect the alternate register set. However, there is no need to know what is in these registers unless your target program first does an EXX (or EX AF,AF'), and later exchanges registers again. By stepping through the first exchange, you will know what has been saved in the alternate registers; MicroMind will not change their contents.

(10) You can't easily change the contents of the Add/Subtract or the Half-Carry flags. By inspecting the AF register (in a full register display), you can at least ascertain the state of these flags. It is highly unlikely that you will need to change that state.

(11) The P1 and P2 displays are updated after each use of the MM command to change memory, and after each EXECUTE cycle in SP mode.

(12) MicroMind initializes a stack for use of the target program; this stack suffices for nearly all debugging. If your program must set up its own stack, be sure that it does not employ any memory that would interfere with the operations of MicroMind or (in the case of integral MicroMind) Instant Assembler; the same remark applies to any storage areas that your program may establish. To avoid contaminating Instant Assembler, do not use any memory locations below the address given by Instant Assembler in its "1ST FREE MEM" report when the AM command is exercised.

(13) MicroMind will disassemble the undocumented instructions, which are defined in item (17) of Section 3.

(14) In entering any command or information, you may use the LEFT ARROW key to backspace and erase characters.

PART III. THE LINKING LOADERS

Linking Loader is a machine language program that will load a module produced with Instant Assembler into any RAM location outside its own program and storage areas. It will also load and link multiple modules. (The modules that Linking Loader operates on are the source code modules recorded with the OS command of Instant Assembler.) In addition, Linking Loader can record an object file of the loaded program.

Linking Loader occupies approximately 3300 bytes at one end of RAM and requires a certain amount of memory (which is dynamically allocated as needed) adjacent to its program area for the storage of labels and their values. Object code is assembled and placed in memory in real time (as the input file is read), so that no buffer space is required for the code itself. In loading multiple modules, Linking Loader proceeds from the specified starting point toward its own storage area. It will stop -- with an "OUT OF MEM" report -- if it runs out of room. In a 32K RAM there should be enough space to load a multi-segment program of at least 16K bytes; the exact upper limit depends upon the sizes of the individual modules, the number of cross references between modules, and even the order in which the modules are loaded.

Linking Loader is supplied in two versions. The Top-Down Loader resides in low RAM and loads programs into high RAM, while the Bottom-Up Loader occupies the top of RAM and loads programs into low RAM. As the names suggest, the Top-Down Loader loads programs downward from a specified top address, and the Bottom-Up Loader loads programs upward from a specified bottom address. The Top-Down Loader has the file name "DSKLLT/CMD"; its entry point is 5800H. Two files on your program diskette contain copies of the Bottom-Up Loader. The copy in the file "DSKLLB32/CMD" loads to the top of a 32K RAM and has an entry point at 0BFFBH; the copy in the file "DSKLLB48/CMD" loads to the top of a 48K RAM and has an entry point at 0FFFBH.

SECTION 7. LINKING LOADER COMMANDS

Linking Loader has eight two-letter commands, which are explained in this section and summarized in Appendix 7. As in Instant Assembler and MicroMind, entry of the second letter of each command triggers the response. The command prompt is "#". The BREAK key can be used at any time except during input/output to return to command level.

LD (LoaD and link source modules)

With Linking Loader running, answer the "#" prompt with "LD". Linking Loader will respond with the legend "LOAD-". After this, the sequences to be followed differ slightly with the two versions of Linking Loader.

(a) The Top-Down Loader will now announce "LAST FREE MEM: XXXX", where the XXXX is either the top of memory (if this is the first module to be loaded), or the last address preceding the last module loaded (if another module has already been loaded). You will then be asked for a "FILE NAME?". Respond with the file name of the module that you want to load next. Linking Loader will then ask for a "FINAL ADDRESS?". Answer this question with the memory address of where you want the module to end; pressing ENTER in response to this query sets this final address to the default value, which is the value announced in the "LAST FREE MEM" report. In any case, Linking Loader will not accept a final

address higher than the default value. After the file name and final address have been entered, the file will be loaded and linked. If the load is error free, Linking Loader will report "GOOD" and then prompt you to load the next module with the same sequence of messages and queries as for the first module. On the other hand, if a module fails to load properly, Linking Loader will report "BAD" and then prompt you to try again to load this module by repeating the sequence of messages and queries given above. Any disk read error will also result in an error message, followed by a prompt to try again. The Top-Down Loader will load successive modules into successively lower memory areas, though these areas do not have to be contiguous. (To make them contiguous, merely press ENTER in response to the "FINAL ADDRESS?" query for each module after the first.)

(b) The sequence for the Bottom-Up Loader is similar, but with a few differences. The report "1ST FREE MEM: XXXX" (instead of "LAST FREE MEM: XXXX") is issued before loading each module after the first; the reported address is one higher than the last address of the last module loaded. (No report is issued for the first module to be loaded.) Next, you will be asked for a "FILE NAME?". After entry of the file name, you will be asked for an "ORIGIN?" (instead of a "FINAL ADDRESS?"). Answer this question with the memory address of where you want the module to start; pressing ENTER in response to this query (for any module after the first) sets the origin to the default value, which is the value announced in the "1ST FREE MEM" report. In any case, Linking Loader will not accept an origin lower than the default value. (There is no default value for the first module; an origin must be entered for it.) After the file name and origin have been entered, loading and linking occur as with the Top-Down Loader; reports on the outcome of the load are also the same as with the Top-Down Loader. When the module has been successfully loaded, you will be prompted to load the next module with the sequence of messages and queries given above. The Bottom-Up Loader will load successive modules into successively higher memory areas, which need not be contiguous. (To make them contiguous, merely press ENTER in response to the "ORIGIN?" request for each module after the first.)

With either version of Linking Loader, after you have loaded the last module, press the BREAK key when the next FILE NAME? is requested. At this point Linking Loader will report all assembly errors that it discovered. This error report has the following format:

```
INT ERRS: 001
EXT UNDEF SYMBS: 002
&MULT    &SRCE
```

Here, "INT ERRS" (for internal errors) gives the total of all errors resulting from undefined nonexternal symbols, relative jumps out of range, and relative jumps with targets that are labels defined externally (that is, not in the same modules as the relative jumps). Linking Loader will not try to link a jump of the last type even if it is within the allowed range of a relative jump. All of these internal errors should have been eliminated before the modules were recorded; they could easily have been found and corrected via the LI command.

"EXT UNDEF SYMBS" gives the number of external symbol references that have no corresponding labels to define them. All these symbols are then listed below the count, eight to a line. (If 12 or more lines of these symbols are to be listed, Linking Loader will pause after each 12 lines; press ENTER to continue the listing.) In the above example, "&MULT" and "&SRCE" should have appeared as

labels in some module (or modules), but did not. It is also possible that these represent misspellings of actual labels, in which case the LE command of Instant Assembler can be very helpful in tracking them down. In any event, you will have to correct these errors eventually.

There is one type of error that Linking Loader will not detect, and that is an external label that appears in more than one module -- a doubly defined external label. In supplying an address for an instruction that references such a label, Linking Loader will use the latest defined value -- that is, the value of the label in the most recently loaded module in which it appears. Thus, an error of this type may or may not result in an actual error in the assembled program. Of course, the way to avoid the possibility of a real error of this type is to use each external label in only one module.

CL (Continue Loading and linking)

Occasionally you may terminate a link-loading operation before it has been completed. One reason might be to see what external symbols remain undefined at a particular point in the loading process. Another might be to check on the values of some external labels that have already been loaded; the SY command, explained below, can be used for this purpose. When you are ready to continue loading, enter the CL command. Linking Loader will resume exactly as if there had been no interruption.

SY (display SYmbol values)

At any time after terminating a load operation, you may use the SY command to learn the load address of any instruction with an external label. After "#SY", you will be asked for a "SYMBOL?". Respond with any external label in the program that has been loaded. (If you name a nonexternal label, or one that does not exist in the program, Linking Loader will reply "BAD" and ask for the symbol again.) Linking Loader will report the absolute memory address (in hex) of the instruction at that label and then ask for another "SYMBOL?". You may thus learn the memory addresses of as many external labels as you please. When satisfied, press BREAK to return to command level.

PM (Print a load Map)

The PM command allows you to print the memory addresses of all external labels in a program that has been loaded. Have the printer turned on and ready. The external labels and their values will be printed four to a line. This listing is not in alphabetic order, though it is in approximately numeric order. With the Top-Down Loader, the labels will be printed in generally descending order, while, with the Bottom-Up Loader, they will be printed in generally ascending order.

OO (Output Object file to disk)

With the OO command you can make an object file of a program that you have loaded and linked with Linking Loader, or record (on disk) any other machine language program that is in memory. The recorded program is in standard object format, ready to be loaded and executed from DOS. A program can be recorded in up to five noncontiguous segments.

After "# 00", Linking Loader will request a "FILE NAME?". Respond with the name you have chosen for the object file. Linking Loader will then say "SEGMENT 1:" and follow this with a request for a "FIRST ADDRESS?". This is the lowest address of the program that you wish to record, and it may be entered in either decimal or hex (as explained in Appendix 5). There is also a default option available for this address: If you are recording a program that you have just loaded with Linking Loader, you may press ENTER in response to the "FIRST ADDRESS?" query, and the correct beginning address will be automatically supplied. Next, Linking Loader will request a "FINAL ADDRESS?". This is the last memory address of the first (perhaps only) segment of the program to be recorded, and it may be entered in either decimal or hex. Again, there is a default option for this address: If you are recording a program that you have just loaded with Linking Loader, you may press ENTER in response to the "FINAL ADDRESS?" query, and the end address (for the complete program) will be automatically supplied. (If you are recording the program in several segments, do not use this default option.) After the first and final addresses have been entered, Linking Loader will ask for an "ENTRY ADDRESS?", which is the address at which execution of the machine language program is to start when later it is called from DOS. An address must be entered here (in either decimal or hex) even if it is meaningless.

After the addresses for Segment 1 have been entered, Linking Loader will ask for first and final addresses for Segment 2, Segment 3, Segment 4, Segment 5. To terminate this request sequence, press ENTER in response to the "FIRST ADDRESS?" query for any segment after the first. When all segment addresses have been entered, DOS will be asked to open the file. Depending upon the outcome of this request, Linking Loader will report "NEW FILE." (followed by transfer of the object code to disk) or "FILE REWRITE. PROCEED (Y/N)?" (which means that a file with this name already exists). In the latter case you then have a choice of continuing or aborting the operation.

If you change any of Instant Assembler's program parameters (using the information in Appendix 6) and wish to record the modified program, use the 00 command of the Bottom-Up Loader. The FIRST ADDRESS for this recording is 5B00 (hex). The FINAL ADDRESS can be found by looking into memory locations 5B06H-5B07H when Instant Assembler is in memory. (Remember that 5B07H contains the high-order byte of this address.) The ENTRY ADDRESS is also 5B00 (hex). Use only one segment for the recording.

The 00 command may also be used to record a relocated version of stand-alone MicroMind. The addresses for this recording are displayed by the relocater at the time of the relocation. If MicroMind is relocated to high RAM, use the Top-Down Loader for the recording; if MicroMind is relocated to low RAM, use the Bottom-Up Loader.

TP (record object code on TaPe)

With the TP command you can make a 500 baud object tape of a program that you have loaded and linked with Linking Loader, or record (on tape) any other machine language program that is in memory. The recorded program is in standard object format, ready to be loaded and executed with the SYSTEM command of Level II. A program can be recorded in up to five noncontiguous segments.

After "# TP", Linking Loader will ask for a "TITLE?"; type in any name of six or fewer characters. Following this entry, Linking Loader will request addresses for the program segments exactly as explained under the 00 command above. After all addresses have been entered, recording will commence. Have the cassette recorder turned on, with the PLAY and RECORD keys depressed.

VF (VeriFY an object tape)

The VF command allows you to verify a 500 baud machine language tape that you have just recorded with the TP command. Rewind the tape to the beginning of the recorded program and type "VF" in response to the "#" prompt. Linking Loader will immediately try to verify the recording, so press the PLAY key on the recorder. If anything is wrong, "BAD" will be displayed, and you may adjust the volume control or re-record the program and try again to verify it. If the recording is all right, "GOOD" will be displayed when the verification is complete.

JP (JumP)

The JP command permits transfer of control to any point in memory. After you have entered this command, you will be asked for an "ADDRESS?". Enter this in either decimal or hex, and the jump will be taken to that address.

SECTION 8. EXAMPLE OF LINKING LOADER IN ACTION

In this final section you will use the two versions of Linking Loader to load and link the three-segment hex-to-decimal conversion program that you constructed in Section 2. The first part of that program calls subroutines in each of the other two parts; thus, Linking Loader will need to determine the addresses of those subroutines as it loads them and then plug those addresses into the calling instructions of Part 1 of the program.

Load and run the Top-Down Loader. Enter the LD command and type in "HDCNV3/SRC" in answer to the "FILE NAME?" query. For a FINAL ADDRESS, enter 90D1. (Be sure the diskette with the hex-to-decimal converter source files is mounted.) Part 3 will then be loaded, and Linking Loader will request another file name. Respond with "HDCNV2/SRC", and press ENTER in answer to the "FINAL ADDRESS?" query. Part 2 will then be loaded, and Linking Loader will request another file name. Respond with "HDCNV1/SRC", and press ENTER in answer to the "FINAL ADDRESS?" query. Part 1 will then be loaded and linked. (The three modules will be loaded contiguously because you used the ENTER key to enter default final addresses for Parts 2 and 1.) Now press the BREAK key when Linking Loader asks for another file name. The error report that appears should show no errors. Enter the SY command, and type "&BEGIN". Linking Loader will then announce the memory address of the first instruction of the hex-to-decimal converter; this address should be 9000. You now see the reason for using an external label at the entry point of the converter; although &BEGIN is not referenced by either Part 2 or Part 3 of the program, it may be referenced by the Linking Loader. Linking Loader cannot give you the value of any nonexternal label. Now press BREAK to return to command level. If you wish, you may use the PM command to print a load map of your hex-to-decimal converter.

Enter the JP command, and transfer control to the converter (JP to 9000). After satisfying yourself that the program has been properly loaded and linked, use the CLEAR key to return to DOS.

The program that you have just loaded could have been located in any region of memory above the Top-Down Loader; you might wish to reload it into another area. (You could even load the three parts noncontiguously, by entering actual FINAL ADDRESSES for all three parts during the loading.) Also, the three modules can be loaded in any order; you might want to repeat the linkage-loading procedure with a different order of loading. However, it is frequently true in top-down loading that one module defines the end of a storage area of indeterminate size that lies below the program area; in such a case, care must be exercised to insure that this module is the last one loaded. There are also circumstances in which one particular module must be the first one loaded. The main fact to keep in mind is that -- in top-down loading -- successive modules are loaded into successively lower regions of memory. (It does not follow, though, that a single module is loaded from higher to lower addresses; indeed, the reverse is true.)

Now is the time to load and run the Bottom-Up Loader. Enter the LD command, followed by a FILE NAME of "HDCNV1/SRC". Give an ORIGIN of 9000. After Part 1 has been loaded, load Parts 2 and 3. (Any other order would work as well.) Terminate the loading process with the BREAK key. Again the error report should show no errors. Use the SY command to learn the entry point address (the value of &BEGIN). Transfer to this entry point with the JP command, and finally return to DOS (from the converter) by pressing the CLEAR key.

In bottom-up loading, successive modules are loaded into successively higher memory locations; keep this fact in mind if the successful operation of the total program is dependent upon the relative positions of the segments in

memory. it should also be clear that, the lower you set the origin for the load, the more memory space there will be for the program that you are loading.

APPENDIX 1. LEGAL INSTRUCTIONS FOR INSTANT ASSEMBLER 2.1

ADC A,s	DJNZ e	JP nn	LD R,A	PUSH IX
ADC HL,ss	EI	JR C,e	LD r,m	PUSH IY
ADD A,s	EQU n3	JR NC,e	LD r,n	PUSH qq
ADD HL,ss	EX (SP),HL	JR NZ,e	LD SP,HL	RES b,m
ADD IX,pp	EX (SP),IX	JR Z,e	LD SP,IX	RET
ADD IY,rr	EX (SP),IY	JR e	LD SP,IY	RET cc
AND s	EX AF,AF'	LD (BC),A	LD ss,(nn)	RETI
BIT b,m	EX DE,HL	LD (DE),A	LD ss,nn	RETN
CALL cc,nn	EXX	LD (HL),n	LD u,x	RL m
CALL nn	HALT	LD (HL),r	LD x,n	RLA
CCF	IM 0	LD (IX+d),n	LD x,u	RLC m
CP s	IM 1	LD (IX+d),r	LD x,x'	RLCA
CPD	IM 2	LD (IY+d),n	LDD	RLD
CPDR	IN A,n2	LD (IY+d),r	LDDR	RR m
CPI	IN r,(C)	LD (nn),A	LDI	RRA
CPIR	INC IX	LD (nn),IX	LDIR	RRC m
CPL	INC IY	LD (nn),IY	NEG	RRCA
DAA	INC m	LD (nn),ss	NOP	RRD
DEC IX	INC ss	LD A,(BC)	OR s	RST p
DEC IY	INC x	LD A,(DE)	OTDR	SBC A,s
DEC m	IND	LD A,I	OTIR	SBC HL,ss
DEC ss	INDR	LD A,(nn)	OUT (C),r	SCF
DEC x	INI	LD A,R	OUT n2,A	SET b,m
DEFB n	INIR	LD I,A	OUTD	SLA m
DEFM 'cs'	JP (HL)	LD IX,(nn)	OUTI	SRA m
DEFS n1	JP (IX)	LD IX,nn	POP IX	SRL m
DEFW nn	JP (IY)	LD IY,(nn)	POP IY	SUB s
DI	JP cc,nn	LD IY,nn	POP qq	XOR s

Operand Notation

b represents a number in the range of 0 to 7.
 d represents a one-byte number in the range of -128 to 127.
 e represents a symbolic address within relative range (-126 to 129).
 n represents any one-byte number or character constant.
 n1 represents a number in the range of 1 to 4095.
 n2 represents a number in the range of 0 to 255.
 n3 represents a number in the range of 0 to 65535.
 p represents one of the following: 0, 8, 10H, 18H, 20H, 28H, 30H, 38H.
 r represents any of the following registers: A, B, C, D, E, H, L.
 m represents either r, (HL), (IX+d), or (IY+d).
 u represents any of the following registers: A, B, C, D, E.
 x represents any of the half-index registers: IXH, IXL, IYH, IYL.
 x' represents any of the half-index registers IXH, IXL, IYH, IYL, with the restriction that x and x' must be halves of the same register.
 s represents either r, n, (HL), (IX+d), (IY+d), or x.
 nn represents a two-byte number or address; it may be a symbol.
 cs represents an ASCII string of not more than 43 characters.
 cc represents any of these conditions: NZ, Z, NC, C, PO, PE, P, M.
 pp represents any of these 16-bit registers: BC, DE, IX, SP.
 qq represents any of these 16-bit registers: AF, BC, DE, HL.
 rr represents any of these 16-bit registers: BC, DE, IY, SP.
 ss represents any of these 16-bit registers: BC, DE, HL, SP.

APPENDIX 2. SUMMARY OF ASSEMBLER COMMANDS

COMPOSING AND EDITING

CP -- ComPose source code
ED -- EDit lines of source code
CC -- Continue Composing

INSERTING, DELETING, MOVING

IS -- InSert lines of source code
DL -- Delete one Line of source code
DM -- Delete Multiple lines of source code
MB -- Move a Block of source code

LISTING

LC -- List Completely (to the screen)
PC -- Print a Complete listing
LL -- List to the Last line (from a specified starting line)
PL -- Print (a listing) to the Last line
PR -- Print a Range of lines
LI -- List Internal errors
PI -- Print Internal errors
LE -- List External undefined symbols
PE -- Print External undefined symbols
LS -- List the Symbol table
PS -- Print the Symbol table

TAPE INPUT/OUTPUT

WS -- Write Source code to tape
VS -- Verify Source code
RS -- Read Source code from tape
WO -- Write Object code to tape
WE -- Write Edtasm source code to tape
RE -- Read Edtasm source, translate, and merge

DISK INPUT/OUTPUT

OS -- Output Source code to disk
IN -- INput source code from disk
MG -- MerGe source code from disk
OO -- Output Object code to disk
OE -- Output Edtasm source code to disk
IE -- Input Edtasm source, translate, and merge

MISCELLANEOUS

AM -- Assemble-to-Memory
RO -- Reset Origin
FR -- Find all References to a specified symbol
DI -- DIrectory display
KL -- Kill a file
EX -- EXit to DOS
MD -- transfer to microMinD

APPENDIX 3. SOURCE CODE ENTRY

EXAMPLES: (Numbers in parentheses below examples refer to the notes.)

(a) Instruction line:

```
0045 &ENT2 LD A,(HL) ;GET CHARACTER
(1) (2) (3)(4)(3) (5) (6) (7) (8)
```

(b) Comment line:

```
0123 ;The next routine converts numeric input.
(1) (7) (8)
```

NOTES:

(1) Four-digit line number and following space are provided by Instant Assembler.

(2) Label is optional and is limited to six alphanumeric characters, except that the first character may be an ampersand to indicate an external label. (If first character entered is a semicolon, line will be converted to a comment line.)

(3) Tab (RIGHT ARROW) to the next field.

(4) Opcode is mandatory for standard line and is limited to four alphabetic characters.

(5) Enter necessary operand or operands in this field. If two operands are required, separate them with a comma. Do not use spaces in the operands (except for a character constant).

(6) Spaces preceding the on-line comment are optional.

(7) The semicolon is required to indicate that a comment follows. Note that in a comment line the starting semicolon cannot be erased except by use of SHIFT-LEFT ARROW.

(8) Comments are free form. Lower case may be used in the Model III. No on-line comment is permitted following a DEFM pseudo-instruction.

COMPLETION MODES:

```
SHIFT-LEFT ARROW -- erase this line and start fresh with the same
                    line number.
BREAK              -- abort this line and return to command level.
ENTER              -- normal end of line; line is checked and (if
                    error-free) entered into source buffer.
```

APPENDIX 4. EDITING PROCEDURES

The ED, LI, and FR commands provide direct editing facilities. The CP, CC, and IS commands permit editing while entering a line; they also lead to editing if the entered line has a detectable error.

LINE LEVEL:

ED command only:

UP ARROW	-- Display previous line.
D	-- Delete displayed line.
I	-- Insert new line before displayed line.

ED, LI, FR commands:

DOWN ARROW	-- Display next line.
ENTER	-- Same as DOWN ARROW.
C	-- Descend to cursor level to edit line.

CURSOR LEVEL (ED, LI, FR, CP, CC, IS):

SPACE	-- Move cursor right one space without erasing.
LEFT ARROW	-- Move cursor left one space without erasing.
RIGHT ARROW	-- Tab to next field; if that field is empty, descend to edit level, X mode.
SHIFT-D	-- Delete character at the cursor.
SHIFT-I	-- Descend to edit level to insert characters.
SHIFT-C	-- Descend to edit level to change characters.
SHIFT-H	-- Delete characters to end of field; descend to edit level to enter characters.
SHIFT-X	-- Tab to end of field; descend to edit level to enter characters
SHIFT-LEFT ARROW	-- Erase entire line and start over.
ENTER	-- Normal termination of editing.
BREAK	-- Same as ENTER, if used with ED, LI, or FR. Abort, if used with CP, CC, or IS.

A one- or two-digit number (n) typed just before SPACE, LEFT ARROW, or SHIFT-D extends the effect of each key over n characters.

EDIT LEVEL (ED, LI, FR, CP, CC, IS):

A character typed at this level is entered into the source line, provided that it is legal for the field of entry.

LEFT ARROW	-- Backspace without erasing in I or C mode. Backspace and erase in H or X mode.
RIGHT ARROW	-- Tab to next field; if that field is not empty, return to cursor level.
DOWN ARROW or SHIFT-UP ARROW	-- Return to cursor level.
SHIFT-LEFT ARROW, ENTER, or BREAK	-- All have the same effect at edit level as they have at cursor level.

APPENDIX 5. ENTERING LINE NUMBERS AND ADDRESSES

LINE NUMBERS:

- (1) Decimal: Enter four or fewer decimal digits.
- (2) Label plus offset: Enter the label of any line in the program. This label may also have a decimal offset in the range of -31 to +99.
- (3) Current line: Enter "." (period) to request the current line. Type UP ARROW to request the line preceding the current line. Type DOWN ARROW to request the line following the current line.

How the current line number is maintained:

- (a) It is initialized at 1 when Instant Assembler is loaded.
- (b) Whenever source code is displayed on the screen, the current line pointer is updated to the number of the last displayed line. Exception: When a line that is being entered (with the CP, CC, or IS command) is aborted with the BREAK key, the current line pointer is not updated to the number of that line.
- (c) When the CC command is entered, the current line pointer is set to the number of the last line in the source buffer.
- (d) Use of the UP ARROW to enter a line number not only selects the line before the current line, but also resets the current line pointer to the number of the selected line. Similarly for DOWN ARROW.
- (e) Printing and block movement have no effect on the current line.

ADDRESSES:

- (1) Hexadecimal: Enter four or fewer hex digits. Do not type a zero before a leading A, B, C, D, E, or F. Do not type "H" at the end of the entry. (NOTE: This rule is for addresses entered in answer to a query; hex numbers entered in source code must have both the leading zero and the terminating H.)
- (2) Decimal: Enter five (no fewer) decimal digits. If necessary, pad with leading zeroes to make the total of five digits.

ORIGINS AND ENTRY ADDRESSES:

Besides the two methods given above for all addresses, origins and entry addresses for Instant Assembler can be entered as default values by simply pressing ENTER. The default value is the last origin set with the RO command, which is also the origin that appears when the program is listed with the LC command.

APPENDIX 6. PARAMETER LOCATIONS AND MEANINGS

5AFEH - 5AFFH (Top of memory) Computed whenever Instant Assembler is entered at 5B00H. You may set this value to protect other programs in high RAM. Subsequently, use the reentry point at 5E03H to avoid destroying the protection.

Changes in any of the parameters below may be made permanent by using the Bottom-Up Linking Loader to record the modified Instant Assembler, as detailed under the 00 command of Section 7.

5B08H (Directory option flag) Comes set at zero for Model III TRS-DOS. Set to 1 for NEWDOS 80 or Model III LDOS 5.1. Set to 2 for DOSPLUS 3.4.

5B09H (Number of spaces of indentation in printed listings) Comes set at 8. If you set bit 7 (the 80H bit) of this number, you can obtain source-only listings. These listings will use 18 fewer print columns than the number you put in 5B0AH.

5B0AH (Number of print columns) Comes set at 72; must be set at a number from 64 to 98, or to 105. (A setting of 105 guarantees that no instruction will take more than a single line.) This number plus the number in 5B09H should not exceed the column capacity of your printer. (Exception: If bit 7 of location 5B09H is set, the sum of these two numbers can be up to 18 greater than your printer's column limit.)

5B0BH (Number of printed lines per page of listing) Comes set at 59.

5B0CH (Total number of lines per page) Comes set at 66. If this number is greater than the number in 5B0BH, the difference will be the number of copies of the character in 5B0DH transmitted at the end of each page. If 5B0CH is set to 1, only one copy of the character in 5B0DH will be sent after each page.

5B0DH (Line feed or form feed character) Comes set at 0AH, which is normally sent several times at the end of each page. If you change 5B0CH to 1, put a form feed in 5B0DH.

5B0EH (Carriage return character or flag) Comes set at 0DH, which is transmitted at the end of each line. If 5B09H - 5B0AH are jointly set to transmit full lines to your printer, and if your printer doesn't require a carriage return in this case, then store a zero in 5B0EH. A zero here will not be transmitted.

5B0FH - 5B14H (Custom print formatting bytes) Come set at all zeroes. Any characters you store here (up to the first zero byte, which will not be transmitted) will be sent to your printer at the start of each line. Thus, you can program compressed format, double strike, etc., if your printer permits.

5B15H (EDTASM source format flag) Comes set at 1 for recording six spaces at the beginning of an EDTASM source file. Set to 0 to suppress this dummy title.

APPENDIX 7. SUMMARY OF MICROMIND AND LINKING LOADER COMMANDS

MICROMIND

STEPPING, BREAKPOINTING, AND EXECUTING

SP -- Step	XC -- eXeCute
BD -- Blank Display	RN -- RuN
BK -- Breakpoint	RB -- Restore Breakpoint
SB -- Step from Breakpoint	JP -- Jump
CL -- Call	

REGISTER AND MEMORY DISPLAY

RG -- ReGister display and change	MM -- MeMory display and change
AS -- AScii display	P1 -- Page 1
P2 -- Page 2	

UTILITIES

FN -- Find Number	DS -- DiSassemble
HD -- Hex-to-Decimal conversion	DH -- Decimal-to-Hex conversion

SYMBOLIC DISASSEMBLY AND TRANSFER (Integral MicroMind Only)

SD -- Symbolic Disassembly	AD -- Absolute Disassembly
IA -- transfer to Instant Assembler	

TAPE AND PRINTER COMMANDS (Stand-Alone MicroMind Only)

TP -- TaPe	VF -- VeriFy
DP -- Disassemble and Print	

LINKING LOADER

LD -- LoAD and link source modules	CL -- Continue Loading and linking
SY -- display SYmbol values	PM -- Print a load Map
OO -- Output Object file to disk	TP -- record object code on TaPe
VF -- VeriFy an object tape	JP -- Jump

APPENDIX 8. ADAPTING TO EDTASM

Since some of EDTASM's constructions are not recognized by Instant Assembler, you may need to make some adaptations when copying programs from magazines, because these generally assume the use of EDTASM. The key to success is to understand the function of each line of code that you are transcribing; a functionally equivalent Instant Assembler form is nearly always available. Some suggestions for these conversions have already been made in Section 3, items 11, 12, 13. In this appendix are more suggestions, illustrated by examples culled from the pages of "80 MicroComputing" magazine. You may notice that many of these examples are awkward constructions even in EDTASM.

(1) The first set of suggestions concerns the use of the symbol "\$" to mean the address of the present instruction. Here are some examples:

(1A) JR NZ,\$+6

SUGGESTION: Put a label -- say "HERE" -- on the instruction at \$+6, then change the above line to JR NZ,HERE. To figure out where \$+6 is, you will need to count the bytes of each instruction; start with the JR NZ,\$+6, which counts for two bytes. After you have counted six bytes worth of instructions, put the label on the next instruction.

(1B) DJNZ \$

SUGGESTION: Change this to TITELP DJNZ TITELP.

(1C) NOTU EQU \$
CP 2AH

SUGGESTION: Replace these two lines with NOTU CP 2AH. The only purpose served by the EQU here is to attach the label to the instruction in the line following; this can be done directly as suggested.

(1D) BOARDS DEFL \$
END

SUGGESTION: Use BOARDS DEFS 1 instead. (Instant Assembler doesn't want the END instruction.) In the actual program, BOARDS was a label defining the beginning of a storage area.

(1E) SET 0,(HL)
OPCODE EQU \$-1

SUGGESTION: Use the following two lines instead:

DEFB 0CBH ;1st byte of SET 0,(HL)
OPCODE DEFB 0C6H ;2nd byte

(1F) ORG OFF00H
DISKIO EQU \$
END

SUGGESTION: Use DISKIO EQU OFF00H instead.

(1G) GETCHR CALL \$-\$

SUGGESTION: Use GETCHR CALL 0 instead, since the value of \$-\$ is zero.

(2) The second set of suggestions concerns the use of arithmetic operators within operands. EDTASM is liberal in its acceptance of these, while Instant Assembler is not. Here are some examples:

(2A) XOR 128+64
SUGGESTION: Use XOR 192 instead.

(2B) LD IX,3C3FH-40H
SUGGESTION: Use LD IX,3BFFH. Figure out the arithmetic yourself.

(2C) LD (IX+256-63),A
SUGGESTION: Use LD (IX-63),A instead. Here we see a minor flaw of EDTASM, which will not accept LD (IX-63),A, though that is the intent of the example instruction. Index register offsets for EDTASM must be in the range of 0 to 255; a negative offset has to be adjusted by adding 256 to it so that EDTASM will accept it. Instant Assembler does this the right way.

(2D) LD (IX+OFFH),A
SUGGESTION: Use LD (IX-1),A instead, which is the correct form. See (2C) above.

(2E) LD HL,CARDIM-36
SUGGESTION: Use the following lines instead:
LD HL,CARDIM
PUSH BC
LD BC,-36
ADD HL,BC
POP BC

This makes the program six bytes longer, which may have to be taken into account when assigning the origin. Be aware that changing the length and origin of a published program can also affect entry point and storage area addresses.

(2F) LD BC,TRCTBE-TRCTAB-1
SUGGESTION: Use the following lines instead:
PUSH HL
LD HL,TRCTBE-1
LD BC,TRCTAB
OR A
SBC HL,BC
LD B,H
LD C,L
POP HL

Again, this makes the program 10 bytes longer, which may have to be taken into account when assigning the origin.

(2G) LD HL,VIDEO+982
SUGGESTION: Use the following lines instead:
LD HL,VIDEO
PUSH BC
LD BC,982
ADD HL,BC
POP BC

Six bytes are added to the code, which may have to be taken into account when assigning the origin. In this particular example, VIDEO was EQUated to 3C00H, so you could also have used the simple replacement line LD HL,3FD6H.

(2H) ADD A,BUFFER<-8
SUGGESTION: Use the following lines instead:

```
PUSH BC
LD BC,BUFFER
ADD A,B
POP BC
```

Once again, four bytes are added to the code, which may have to be taken into account when assigning the origin. In this particular example, BUFFER was EQUated to 6000H, so you could also have used the simple replacement line ADD A,60H.

(3) The third set of suggestions concerns minor variations in the form of operands. Here are some examples:

(3A) LD (DCB+0AH),HL

SUGGESTION: Use LD (DCB+10),HL instead. Instant Assembler requires symbol offsets to be in decimal.

(3B) IN A,(0EEH)
OUT (0EEH),A

SUGGESTION: Use IN A,0EEH and OUT 0EEH,A instead. Instant Assembler requires the omission of the parentheses in these two instructions.

(3C) JEHMN2: LD HL,START

SUGGESTION: Use JEHMN2 LD HL,START instead. Instant Assembler does not permit the colon in the label field.

(4) The fourth set of suggestions concerns the use of the DEFL pseudo-op and an EQU with a symbolic operand. Here are some examples:

(4A) DCB DEFS 32H
EOF EQU DCB+8
ERN EQU DCB+12
NRN EQU DCB+10

SUGGESTION: Use the following lines instead:

```
DCB DEFS 8
EOF DEFS 2
NRN DEFS 2
ERN DEFS 38
```

This gives a total defined storage of 50 bytes and preserves the spacing of the labels.

(4B) DSPDIR DEFL 4419H

SUGGESTION: Use DSPDIR EQU 4419H instead.

(4C) FBUFF DEFL 5200H
START DEFL FBUFF+32H
FINISH DEFL START+2

SUGGESTION: Use the following lines instead:

```
FBUF    EQU    5200H
START   EQU    5232H
FINISH  EQU    5234H
```

(5) The fifth set of suggestions concerns the use of symbols to represent 8-bit (or smaller) operands. Here are some examples:

(5A) DEFB CR

SUGGESTION: Use DEFB 0DH instead, since CR (for "carriage return") had the value 0DH in this example. (The value of the symbol can be determined from the program listing; it is displayed to the left of the instruction.)

(5B) SET FLAG,(HL)

SUGGESTION: Use SET n,(HL) instead, where n is the actual value of FLAG, which is an EQUated symbol.

(5C) OUT (PORT1),A

SUGGESTION: Use the following lines instead:

```
DEFB 0D3H               ;OUT instruction
DEFW PORT1              ;Port number
```

This coding lengthens the program by one byte, which is a NOP corresponding to the zero in the high order byte of the value of PORT1. Another solution is to EQUate a two byte symbol to the entire instruction:

```
OUTP1 EQU 05D3H               ;OUT PORT 5
```

The high order byte of OUTP1 is the port number (05) and the low order byte is the OUT instruction (D3). When you want to use this instruction in your program use OUTP1 as a DEFW:

```
DEFW OUTP1               ;OUT (PORT1),A
```

This coding does not lengthen the program.

(6) The sixth set of suggestions concerns the use of multiple origins. One way to circumvent this difficulty is to construct the program in multiple (up to 5) segments and then load, link, and record it with Linking Loader (which permits up to five origins in the recording of the object code). This might require making some labels external so that they can be referenced by other modules. Another solution to the multiple origins problem applies when the segments are nearly contiguous: Place appropriate amounts of storage (using the DEFS pseudo-op) between segments, thus turning the whole into a single block of code. A third solution may be feasible when some of the segments contain only a jump instruction. Here are some examples:

```
(6A)               ORG 16804-1
                   JP LINE
                   ORG 16762-1
                   JP FIELD
```

SUGGESTION: Instead of these lines, add the following code to the beginning of the actual program:

```
LD    A,0C3H
LD    (16803),A
LD    (16761),A
LD    HL,LINE
LD    (16804),HL
LD    HL,FIELD
LD    (16762),HL
```

This will make the program longer, of course.

```
(6B)      ORG    41E2H
          JP     AUTO
```

SUGGESTION: Put this in a separate module and use Linking Loader to link and record the total program. The reason for a different treatment here is that the lines shown have a special effect in a Model III tape machine. They cause auto-execution of the assembly language program when it is loaded using the SYSTEM command.

```
(6C)      START  EQU    7EAOH
          ORG     START-10
          LD      HL,START
          LD      (418FH),HL
          JP      66H
          ORG     START
          START   CALL   1C9H
```

SUGGESTION: Replace this mess with the following lines:

```
LD      HL,START
LD      (418FH),HL
JP      66H
START   CALL   1C9H
```

To give START the value 7EAOH, use an origin of 7E97H.

INDEX

Topics in this index are referred to not by page number, but by section number and, usually, either a command name or an item number (or both) within the section. The specification "(intro)" refers to the introductory paragraph (or paragraphs) of a section or part. EXAMPLES: 1.1 (CP, item 4) refers to the fourth numbered item under the CP command in Section 1.1, while Part III (intro) refers to the introductory paragraphs of Part III. Besides making it easier to avoid errors in the index, this manner of reference generally pinpoints an item to within less than a printed page.

Alternate registers: 6 (item 9)

Ampersand: 1.1 (CP, item 1), 1.3 (LI), 3 (item 10), Appendix 3 (Note 2)

Assembly-to-memory: 1.6 (AM)

Backspace (LEFT ARROW): 1.1 (CP, items 1 and 4), 1.1 (ED), 6 (item 14),
Appendix 4

Block movement: 1.2 (MB)

BREAK: 1.1 (CP, item 15), 1.1 (ED), 1.2 (IS), 1.3 (intro, LI),
1.4 (intro, RE), 1.5 (intro, MG), 1.6 (FR), 3 (items 1, 5, 14),
4 (intro), 4.1 (SP, BD, RN, JP), 4.2 (RG, MM, AS, P1),
4.3 (FN, DS, HD), 4.5 (DP), 6 (item 4), 7 (intro, LD, SY),
Appendix 3, Appendix 4, Appendix 5

Breakpoints: 4.1 (BK, RB, SB)

Cancel (I, C, H, or X mode): 1.1 (ED), Appendix 4

Character constants: 1.1 (CP, item 8)

Change character: 1.1 (ED), Appendix 4 directory option: Appendix 6
EDTASM source format: Appendix 6 line: 1.1 (ED), Appendix 4
memory: 4.2 (MM) origin: 1.1 (CP, item 5), 1.6 (RO)
print parameter: Appendix 6 register: 4.2 (RG)
top of memory: Appendix 6

C mode: 1.1 (ED), Appendix 4

CODE ERASURE warning: 1.1 (CP), 1.4 (RS, RE), 1.5 (IN), 3 (item 14)

Command summaries: Appendix 2, Appendix 7

Comments full line: 1.1 (CP, item 4), Appendix 3
on-line: 1.1 (CP, items 2, 3), Appendix 3

Conversions: 4.3 (HD, DH)

Current line: 1.1 (ED, CC), 1.2 (MB), 1.3 (intro), Appendix 5

Default addresses: 1.4 (WO), 7 (LD, OO), Appendix 5

Deletion character: 1.1 (ED), Appendix 4
line: 1.1 (ED), 1.2 (DL, DM), Appendix 4

Directory call: 1.6 (DI)

Disassembly absolute: 4.4 (AD) of undocumented instructions: 6 (item 13)
printed: 4.5 (DP) symbolic: 4.4 (SD)

Displays ASCII: 4.2 (AS) memory: 4.2 (MM)
page: 4.2 (P1, P2), 6 (item 11)
register: 4.1 (SP, CL), 4.2 (RG, MM), 6 (item 8)

Doubly defined label: 1.1 (CP, item 16), 3 (items 6, 10), 7 (LD)

DOWN ARROW key: 1.1 (ED), 1.3 (intro, LI), 1.6 (FR), 4.2 (MM, AS, P1),
4.3 (DS), Appendix 4, Appendix 5

Object code, recording of: 1.4 (WO), 1.5 (OO), 4.5 (TP), 7 (OO, TP)
 Offsets index register: 1.1 (CP, item 7) label: 1.1 (ED), Appendix 5
 symbol: 1.1 (CP, item 6)
 Opcode field: 1.1 (CP, item 1), Appendix 3
 Operand field: 1.1 (CP, item 1), Appendix 3
 ORG: 1.1 (CP, item 5), 1.3 (LC)
 OUT OF MEM report: 1.4 (RE), 1.5 (MG), 1.6 (AM), 3 (item 3),
 Part III (intro)
 Out of range errors: 1.1 (CP, item 16), 1.3 (LC, LI)

Parameters, changeable: Appendix 6
 PERIOD key: 1.1 (ED), 1.3 (intro), Appendix 5
 Prompts, command: 1 (intro), 4 (intro), 7 (intro)

Relative jump, target of: 1.1 (CP, item 13)
 Relocation of stand-alone MicroMind: 6 (item 1)
 Restrictions adapting to EDTASM: Appendix 8
 DEFM: 1.1 (CP, items 3, 11)
 DEFS: 1.1 (CP, item 10)
 EQU: 1.1 (CP, item 12)
 module size: 3 (item 3)
 relative jump: 1.1 (CP, item 13)
 symbol: 1.1 (CP, item 6), 3 (items 11, 12, 13)

Single-stepping: 4.1 (SP, BD)
 Source code format: 3 (item 18) size restrictions: 3 (item 3)
 Stack use: 6 (item 12)
 Symbols dead: 3 (item 4)
 restrictions on: 1.1 (CP, item 6), 3 (items 11, 12, 13)
 undefined: 1.3 (LC), 7 (LD)

Tab (RIGHT ARROW): 1.1 (CP, items 1, 9), 1.1 (ED), Appendix 3, Appendix 4
 Top of memory: 3 (item 19), Appendix 6
 Transfer of control: 1.6 (EX, MD), 3 (item 19), 4.1 (JP), 4.4 (IA),
 7 (JP)

Undocumented instructions: 3 (item 17), 6 (item 13), Appendix 1
 UP ARROW key: 1.1 (ED), 1.3 (intro), 4.2 (MM, AS, P1), 4.3 (DS),
 Appendix 4, Appendix 5

Verification of object tape: 4.5 (VF), 7 (VF) of source tape: 1.4 (VS)

X mode: 1.1 (CP), 1.1 (ED), Appendix 4

